May 2009

Manipulations in MAPLE

Matthew Daniel Dailey
Worcester Polytechnic Institute

Project Number: CS-MKH 2008

Manipulations in MAPLE

A Major Qualifying Project Report

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Matthew D. Dailey

Date: May 1, 2009

Approved:

Professor Micha Hofri, Major Advisor

Abstract

The symbolic programming language MAPLE uses different algorithms to find closed forms for symbolic indefinite summations. These algorithms do not explicitly contain a procedure to interchange the order of summation. As a result, MAPLE will return the original summation or a small variation of it when simply interchanging the order of summation would allow for a closed form expression to be found. In this project we look into the task of interchanging the order of summation over a specific domain, and provide a MAPLE procedure which automates this task. We then look into identities involving harmonic numbers which we show through this new procedure, existing MAPLE summation techniques and user manipulation. We then analyze the well known algorithm Quicksort using these identities. Lastly, we look into the summation procedure by Moenck which is used by MAPLE for summations of rational functions.

Contents

Chapter 1

Introduction

This project aimed to analyze the methods of summation of expressions which arise from the complexity analysis of algorithms, and to create tools to help perform complex summations with MAPLE. These summations often contain the harmonic numbers in some form, and thus particular attention was paid to these functions. Moreover, since MAPLE often cannot solve simple summations involving these functions, the method of interchanging the order of summation was analyzed. This process was automated on a restricted domain. Using this new procedure, identities are developed which are later used in the analysis of the Quicksort algorithm, which allow for the calculation of the exact number of expected comparisons. It is shown that the manipulations needed for calculating the expected value extend to more complicated summations, such as those which appear in calculating the variance of Quicksort.

The primary interest of this project laid in automating the process of interchanging the order of summation of simple summations. This process can easily be performed by hand, but is not attempted in MAPLE, since the general problem of switching the order of summation is highly dependent upon the assumptions on the bounds of summation. By assuming certain conditions on these bounds to those summations which are of the form (1.1), we are able to automate this procedure. We then provide evidence of the new capability of the summation package in MAPLE, when augmented with this procedure. We first formalize this problem, and then describe our solution. To demonstrate our solutions' capabilities, we work through common summations from the Quicksort algorithm. Lastly, we turn our attention to MAPLE's method to sum rational functions, and provide an illustrative example of this method.

We are interested in interchanging the order of summation in summations which are of the form (1.1).

$$\sum_{outerVar=outerInitial}^{outerFinal} \sum_{innerVar=a\cdot outerVar+b}^{c\cdot outerVar+d} funct(outerVar, innerVar) \tag{1.1}$$

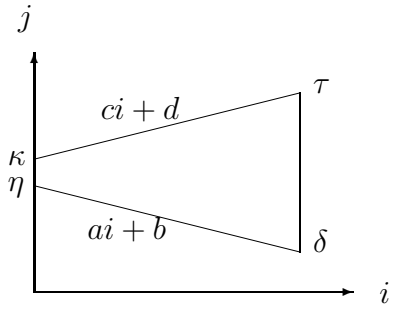
For brevity, we will use i to refer to *outerVar* and j to refer to *innerVar*. We denote the initial value of a variable m by m_o and the final value by m_f. These will be used for i and j.

We will also use f to refer to funct. We will refer in this paper to the line $c \cdot i + d$ as the upper boundary line, and the line $a \cdot i + b$ as the lower boundary line. The area to

be summed over is between these two lines.

$$\sum_{i=i_o}^{i_f} \sum_{j=a\cdot i+b}^{c\cdot i+d} f(i,j) \tag{1.2}$$

The following diagram illustrates summations of the form (1.2).



In the diagram, the point marked by τ is referred to as *finalUpper*, which is the point $(i_f, c \cdot i_f + d)$.
The point marked by δ is referred to as *finalLower*, which is the point $(i_f, a \cdot i_f + b)$.
The point marked by κ is referred to as *initUpper*, which is the point $(i_o, c \cdot i_o + d)$.
The point marked by η is referred to as *initLower*, which is the point $(i_o, a \cdot i_o + b)$.

We will use the convention of Cartesian coordinates with i referring to the X axis and j referring to the Y axis. Hence, we will often refer to lines of the form $y = m \cdot i + \beta$.

In switching the order of these summations, we are really only interested in the relationship between a and c as they determine the general shape of the area which we will be summing over. We are interested in b and d to provide greater insight into the general shape defined by the two boundary lines. We will make no assumptions on f other than that it be defined over the range of the initial summation. We leave it to the user to know if interchanging the order of summation will alter the value of the summation, but note that for finite summations, the order does not matter.

1.1 Purpose

The primary purpose of my work was to automate the process of interchanging the order of summation for a double summation in MAPLE. This was motivated by the existence of many simple summations which could be solved by a computer programing language such as MAPLE if the order of summation was interchanged. The idea was to use this software as a subroutine to a procedure such as MAPLE's *sum*, which would find intermediate results if the summation procedure did not return a closed form expression. Since many of MAPLE's functions are proprietary, my procedure would need to be part of a larger software placed on top of the existing summation function, that would call both my procedure and the summation procedure as subroutines.

This work aimed to aid in the finding closed form expressions for summations which arose out of the analysis of algorithms. These expressions heavily emphasized summations involving the harmonic numbers, which are emphasized in later sections. As non primary goals, I was to become familiar with and proficient at MAPLE as a programming language,

to understand the way MAPLE stores functions and other data structures, and to learn some of the ways in which MAPLE manipulates those structures. I looked at the different algorithms MAPLE uses to find closed form expressions for indefinite summations, which I present in a later section. I studied in greater depth Moenck's Method and a similar algorithm due to Paule.

A test for my procedure was to work through finding the expected value and variance of the Quicksort algorithm when given a paper which set up the appropriate summations with the end result, but which lacked the intermediate steps. My procedure was to be used in conjunction with manual manipulation and built in MAPLE procedures to arrive at the closed forms presented in this paper. Many of the identities used in the manual manipulation involve harmonic numbers and are presented later. Most of their closed form expressions are found by interchanging the order of summation.

Chapter 2

Limitations and Problems

We now discuss some of the initial problems encountered in developing this software. The problems primarily involved early evaluation of expressions, and weakly typed variables. These caused additional checks on the type of variables to be performed prior to their comparison. We also discuss the limitations on the procedure, and state the reasoning for such design decisions. The name of the procedure which we implemented is *sumOrderChange*. It is passed a double summation which is enclosed in single quotes.

2.1 Limitations on Input

2.1.1 Parameter as a double summation

To allow for easier parsing of the double summation whose order is to be interchanged, a restriction is placed on the parameter which can be passed to *sumOrderChange*. If the parameter entered is not a double summation surrounded by single quotes, the input parameter will be returned. This was to stop MAPLE from evaluating the input parameter. The decision to use single quotes tells MAPLE not to evaluate the expression as it is being passed to the procedure. Using the *Sum* function, which is the inert for of the *sum* function, would also accomplish the desired outcome. However, using the inert form *Sum* would require the summation to be parsed and reassembled for my subroutine to be called.

We now look at the restriction of only allowing a double summation to be given to the procedure. We now explain why this decision was made. If there is only one summation being entered, there is no need to change the order. If there are multiple summations, it is unclear how to interchange the order of summation, since there can be different ways to switch the order. It should also be stated that the double summation cannot be a sub expression. This functionality can be added by parsing the input until the first double summation is encountered, and returning the interchanged summation with the constant values added. These 'constant values' can even be other functions of variables, as each parsed input can be compared with the *sum* procedure name.

2.1.2 Linear bounds

The bounds on the inner summation must be either constant or linear, and cannot include the floor function or the ceiling function. Linear bounds which have $a, c \in \pm 1, 0$ avoid using these functions, which I found empirically to improve finding closed forms using MAPLE. Also, linear bounds are easily invertible and lead to countinuos ranges being summed over when the order is interchanged. If this restriction is not placed on the range, continuous ranges of summation may not always be the case.

For example, we considered the case of using a particular third degree polynomial for the lower boundary line in a double summation. When interchanging the order of summation for one such example, we found the following discountiuos range. When j was equal to 10, i was equal to all elements in $1, 2, 3, 10, 11, ..., n-1, n$. However, it is clearly seen that this set is not a continuous range, and thus this interchanged summation must first sum the function for i from 1 to 3 and then from 10 to n.

2.2 Problems encountered

2.2.1 Summations being evaluated

An initial problem we faced with MAPLE was having the summation either be evaluated, or be partially evaluated when it was sent to the procedure. Therefore, the original double summation sent to *sumOrderChange*, may be received in a evaluated form, and may not even contain a solution. We discovered two possible solutions to fix this problem. The first was to use the inert form of the *sum* command, *Sum*. This tells MAPLE not to evaluate the parameter, which in our case is the double summation. This solution was not selected since if this procedure was to be used as a subroutine, the calling procedure would need to replace both *sum*'s with their inert form. This requires parsing the summation in a calling program.

The solution which was used was to have MAPLE treat the double summation as a symbol by enclosing it in single quotes. When the procedure received this symbol, it would not be evaluated, regardless of whether *sum* or *Sum* was used in the summation. We later extended this idea of using quotes to delay evaluation by putting single quotes around each of the *sum* in certain return statements, which would return an unevaluated expression while still simplifying the bounds on the summations. If the interchanged summation was returned from my program while the whole expression was enclosed in single quotes, the bounds on j and i would not be simplified. Placing quotes just around *sum* gave the desired outcome.

2.2.2 MAPLE behavior

A second problem which initially appeared was an unexpected and unexplained behavior by MAPLE when the bounds of a summation went from high to low. Specifically, we will examine bounds which go from

$$(\alpha + k)...\alpha, k \geq 0 \tag{2.1}$$

with summations which take the form of these bounds.

$$\sum_{i=\alpha+k}^{\alpha} f(i)$$

If $k = 0$ then $f(\alpha)$ is returned. If $k = 1$ then 0 is returned regardless of the function f. If $k \geq 2$ then the following is returned.

$$-\sum_{i=\alpha+1}^{\alpha+k-1} f(i)$$

This behavior was initially discovered through certain summations with $A > 0$ and $C < 0$. It quickly became a desired behavior for handling certain types of summations. The purpose of this behavior is to make the following formula hold for all m.

$$\sum_{i=1}^{n} = \sum_{i=1}^{m-1} + \sum_{i=m}^{n} \tag{2.2}$$

Chapter 3

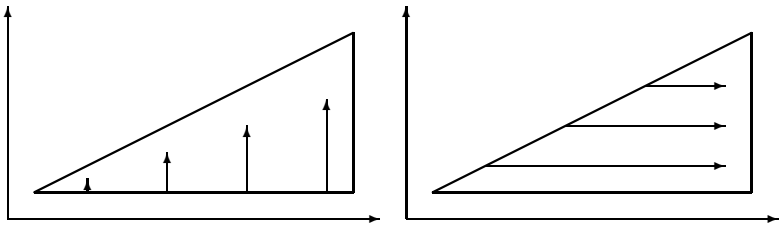
Implementation in MAPLE

In this chapter, we will examine the implementation used to interchange the order of summation in MAPLE. We will briefly cover the procedure to parse the input and extract the necessary elements. We will then examine the procedure corresponding to summing a function over a triangular area, and then the procedure corresponding to summing a function over a rectangular area. We will then discuss in detail the procedure called *DecideCase* which describes how to determine which particular shape the summation area is, and how to correctly interchange the order of summation using the *sumTriangleB* and *sumRectangle* functions where appropriate.

3.1 Parsing the input

A necessary condition for interchanging the order of summation of a double summation is knowing the bounds are of each of the summations. This requires parsing the input to extract these fields. MAPLE uses layered representations of expressions, meaning to get to a particular field, such as the bound on the inner summation, many different structures needed to be indexed.

3.2 Sum Triangle



Sum triangle is a procedure which is given information about a summation over a triangular region. The restriction is that the base of the triangle, which may be at the top or the bottom of the triangle, must be horizontal. The height of the triangle must be a boundary line which is either to the right or to the left of the figure. In other words, the triangle must have a right angle, a horizontal line of slope zero, and a vertical line with infinite slope. These two lines along with a diagonal line define the triangle. The calling of this procedure is done through
sumTriangleB (i, i_o, i_f, j, slope, offset, f, isHorUpper, isVertLeft)

The parameter $isHorUpper$ corresponds to the base of the triangle. $isHorUpper$ is true when the horizontal boundary line is at the top of the triangle, and $isHorUpper$ is false if the horizontal boundary line is at the bottom (base) of the triangle.Similarly, $isVertLeft$ is true if the vertical boundary line of the triangle is to the left, and $isVertLeft$ is false if the vertical boundary line is to the right of the triangle. The decision to include both $isHorUpper$ and $isVertLeft$ is to simplify the procedure and also since the calling procedure should have additional information about the sum in question. We note that it is possible to determine $isVertLeft$ given the slope of the diagonal line and $isHorUpper$.This procedure switches the order of summation from the inner summation being over j, to the inner summation being over i. We briefly examine how this is accomplished. Since the area to be summed over is a triangle, as i ranges from i_o to i_f, we have that j will either range from $slope \cdot i_o + offset$ to $slope \cdot i + offset$ or from $slope \cdot i + offset$ to $slope \cdot i_o + offset$ depending upon the orientation of the triangle.

After interchanging the order of summation the outer index of summation will be j. Therefore, we will need i to range from either $\frac{j-offset}{slope}$ to i_f for a triangle with a vertical line at $i = i_f$, or from i_o to $\frac{j-offset}{slope}$ when the vertical line is at $i = i_o$. We notice that these ranges have integer bounds if $slope$ is either 1 or -1, and therefore, we may need to take either the floor function or the ceiling function of the bounds on these ranges.

Example

For example, we will take $slope = 2$ and $offset = 0$. The triangle will have a horizontal line at the base and a vertical line at the right. Let the value of i range from 0 to 10. Now let us look at the particular cases when $j = 10$ and $j = 11$. When $j = 10, i = 5$ and we wish to sum over values of i from 5 to the boundary line on the right at $i = 10$. When $j = 11, i = 5.5$ and we need to determine if we wish to sum over values of i from 5 to 10, or from 6 to 10. An inspection reveals the point corresponding $(5, 11)$ is not included in the range of summation. Therefore, we wish to sum from 6 to 10 or more generally, from $\left\lceil \frac{j-offset}{slope} \right\rceil$ to 10. We remark that a similar analysis is needed for the other three cases.

We let $outerMin$ and $outerMax$ correspond to the minimum and maximum values possibly dependent upon j and possibly involving the ceiling and floor functions. We also let $innerMin$ and $innerMax$ correspond to the minimum and maximum values ,respectively j can take. We then can return a concise statement to interchange the order of summation. If f is the function we are summing over, then we return
sum(sum(f, i = outMin..outMax),j = inMin..inMax)

Non integer bounds of summation

The *sumTriangleB* function is capable of being passed non integer parameters for the bounds of the outer variable of summation. If the denominator of the parameter divides the slope of the triangle, then this behavior is predictable. There will not be uncertainty if MAPLE will take the floor or ceiling function on the bounds of the resulting summation. This behavior can be seen by seeing that the innerMin and innerMax parameters of *sumTriangleB* are multiplied by the slope a that is passed in. Therefore if, $outerFinal = \frac{N}{a}$ then $a \cdot outerFinal = N$ which is an integer. This behavior is taken advantage of in

certain cases in the procedure *DecideCase*, especially when i is solved for when one of the boundary lines intersects a horizontal line.

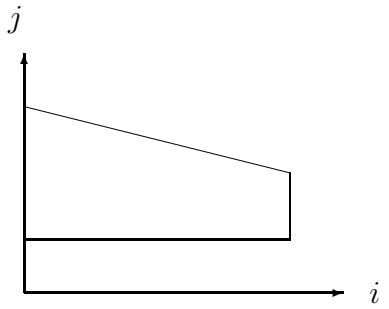
3.3 Sum Rectangle

The procedure *sumRectangle* returns the the order of summation reversed when the area to be summed is a rectangle. The procedure does not perform error checking to make sure the summation is from α to $\alpha + k, k \geq 0$ to take advantage of MAPLE's behavior concerning bounds on summations. The inputs to the procedure are the initial and final values i achieves, as well as the initial and final values j achieves. Additionally the function which is summed is also a parameter. The procedure returns
sum(sum($f, i = i_o..i_f$), j = inLower..inUpper)

3.4 Decide Case

DecideCase is the procedure which takes in the parsed input and decides the general shape of the area to be summed over. It then calls a combination of *sumTriangleB* and *sumRectangle* with the appropriate parameters. This combination and parameters are dependent upon the slopes a, c of the boundary lines. We now provide a detailed analysis for each of the nine combinations of a and c.

3.4.1 $A = 0$ and $C < 0$



We look at the case in which $a = 0$ and $c < 0$. This case corresponds to a triangular region with the horizontal line at the base and the vertical line at the left, with a rectangular region, possibly empty, below this triangle. Here we will add a bound onto this triangle by the assumption that $c \cdot i + d \geq a \cdot i + b$ for all values of i in range.

Informally, the base of the triangle cannot exist below the lower boundary line, $a \cdot i + b = b$ by the assumption that $a = 0$. Therefore, we will apply this limitation to the triangle and hence also the rectangle. The purpose is to reduce the number of rectangle summations which result with sum zero (bounds from $\alpha + 1$ to α), or a summation which will correspond to a range, $\alpha + m$ to α where $m > 1$. As mentioned before, MAPLE will subtract part of this summation from the total.

To see if the two lines could intersect, we compare the two boundary lines, $a \cdot i + b$ with $c \cdot i + d$. More specifically we examine their difference, as b or d are often variables.

theDiff

With the assumption that $a = 0$, we define

$$\textit{theDiff} = (c \cdot i_f + d) - b$$

We will now examine the possible values *theDiff* can take.

theDiff equal to zero

If $\textit{theDiff} = 0$ then we have that $c \cdot i_f + d = b$, which corresponds to the two boundary lines meeting at the point b, when $i = i_f$. We therefore have just a triangle, with horizontal lower base at the line $y = b$ and vertical line on the left at the line $x = i_o$.

In the procedure terms, we return

$$sumTriangleB(i, i_o, i_f, j, c, d, f, false, true)$$

theDiff greater than zero

If $\textit{theDiff} > 0$ then we have that $c \cdot i_f + d > b$, which means there is a triangle with a rectangle of non trivial bounds below it. More specifically, the triangle has horizontal lower base at the line $y = (c \cdot i_f + d)$ and vertical line on the left at the line $x = i_o$. The rectangle is surrounded by the four lines, $x = i_o$, $x = i_f$, $y = b$, $y = (c \cdot i_f + d) - 1$. The -1 is since bounded points of summation are given to triangles over rectangles, as through my analysis, more closed forms arose out of this decision. The line $y = b$ is equivalent to the line $y = \textit{initLower}$. We note that the above case, $(theDiff = 0)$ can be handled the same way as this current case, since the sumRectangle function will return sum zero.

In the procedure terms, we return
sumTriangleB(i, i_o, i_f, j, c, d, f, false, true)+
sumRectangle(i, i_o, i_f, j, initLower, finalUpper - 1, f)

theDiff less than zero

If $\textit{theDiff} < 0$ then we have that $c \cdot i_f + d < b$, which strictly speaking would violate the assumptions on the boundary lines. However, if the difference is one, MAPLE returns sum 0. We will assume a smart user and will return the area where the assumption on the two boundary lines hold. We must now define the point at which these two lines meet, which will be considered the greater value that i attains on the interval. Define

$$\textit{outerMax} = \lfloor \frac{b - d}{c} \rfloor$$

which corresponds to the value of i in $c \cdot i + d = b$. We then sum the triangle for all values of $i \leq \textit{outerMax}$. The claim is that there will not be any points missed or extra points included by *outerMax* being a dependent upon the floor function. We show this later and return to the current case. In this case we return just the triangle with horizontal lower base at line $y = b$ and vertical line on the left at the line $x = i_o$. The value of the area corresponding to $i > \textit{outerMax}$ will be assumed to be zero, which is justified by the assumption of a smart user.

In the procedure terms, we return
sumTriangleB(i, i_o, outMax, j, c, d, f, false, true)

Proof points are not missed

We set $m := \frac{(b-d)}{c}$. To make sure area is not missed, we examine two subcases. In the first, m is an integer, and in the second, m is not an integer.

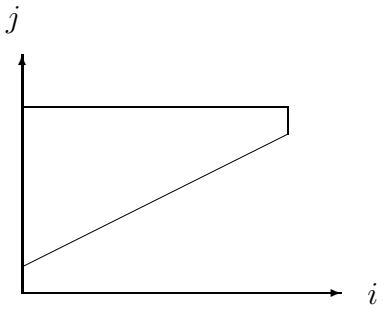
m is an integer

If m is an integer then it is immediate that no area will be missed as $c < 0$ and hence for values of i greater than m, the upper boundary line will be below the lower boundary line. In other words, if i increases by 1 and $c < 0$, then when $cm+d = b$, then $c(m+1)+d < b$.

m is not an integer

By the definition of outerMax, it is claimed there will be no area missed in which $i > outerMax$. This is shown by $c \cdot outerMax + d > b$, as the upper boundary line has a negative slope. Also, $c \cdot (outerMax + 1) + d < b$, since the upper boundary line is linear and the intersection value m is such that $outerMax < m < (outerMax + 1)$. Therefore, outerMax is a proper stopping value for i.

3.4.2 $A > 0$ and $C = 0$



We examine the case in which $a > 0$ and $c = 0$. This case corresponds to a triangular region with the horizontal line at the top and the vertical line at the left, with a rectangular region, possibly empty, above this triangle. Here we will add a bound onto this triangle by the assumption that $c \cdot i + d \geq a \cdot i + b$ for all values of i in range.

Informally, the top of the triangle cannot exist above the upper boundary line, $c \cdot i + d = d$ by the assumption that $c = 0$. Therefore, we will apply this limitation to the triangle and hence also the rectangle. The purpose is to reduce the number of rectangle summations which result with sum zero (bounds from $\alpha + 1$ to α), or a summation which will correspond to a range, $\alpha + m$ to α where $m > 1$. As mentioned before, MAPLE will subtract part of this summation from the total.

To see if the two lines could intersect, we compare $a \cdot i + b$ with $c \cdot i + d$. More specifically we examine their difference, as b or d are often variables.

theDiff

Define

$$theDiff = (a \cdot i_f + b) - d$$

where $initUpper = a \cdot i_f + b$ and $initUpper = c \cdot i_o + d$ which is just d since $c = 0$. We will now examine the possible values *theDiff* can take.

theDiff equal to zero

If $theDiff = 0$ then we have that $a \cdot i_f + b = d$, which corresponds to the two boundary lines meeting at the point d, when $i = i_f$. We therefore have just a triangle, with horizontal upper base at the line $y = d$ and vertical line on the left at the line $x = i_o$.

In the procedure terms, we return
sumTriangleB(i,i_o, i_f, j, a, b, f, true, true)

theDiff less than zero

If $theDiff < 0$ then we have that $a \cdot i_f + b < d$, which means there is a triangle with a rectangle of non trivial bounds above it. More specifically, the triangle has horizontal upper base at the line $y = (a \cdot i_f + b)$ and vertical line on the left at the line $x = i_o$. The rectangle is surrounded by the four lines, $x = i_o$, $x = i_f$, $y = d$, $y = (a \cdot i_f + b) + 1$. The $+1$ is since bounded points of summation are given to triangles over rectangles, as through my analysis, more closed forms arose out of this decision. The line $y = d$ is equivalent to the line $y = initUpper$.

In the procedure terms, we return
sumTriangleB(i, i_o, i_f, j, a, b, f, true, true) +
sumRectangle(i, i_o, i_f, j, initUpper, finalLower + 1, f)

theDiff greater than zero

If $theDiff > 0$ then we have that $a \cdot i_f + b > d$, which strictly speaking would violate the assumptions on the boundary lines. However, if the difference is one, MAPLE returns sum 0. We must now define the point at which these two lines meet, which will be considered the greater value that i attains on the interval.

Define

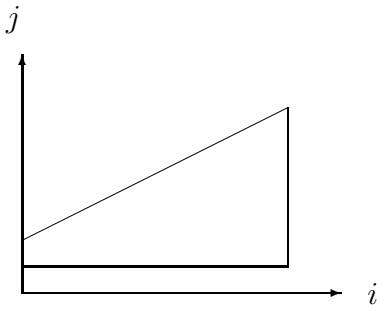
$$outerMax = \lfloor (d - b)/a) \rfloor$$

which corresponds to the value of i in $a \cdot i + b = d$. See the case $A = 0$ and $C < 0$ for a proof of the correctness of *outerMax*.

Here we will return just the triangle with horizontal upper base at line $y = d$ and vertical line on the left at the line $x = i_o$. The value of the area corresponding to $i > outerMax$ will be assumed to be zero, which is justified on the condition of the boundary lines.

In the procedure terms, we return
sumTriangleB(i, i_o, outerMax, j, a, b, f, true, true)

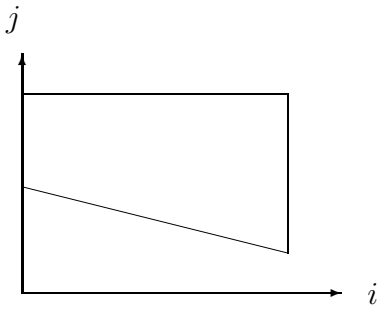
3.4.3 $A = 0$ and $C > 0$



We now examine the case in which $a = 0$ and $c > 0$. Here we have a lower rectangular region and a triangular region with horizontal lower base and vertical line to the right. We return the triangle region and the rectangle region, with the convention of giving the line $y = initUpper$ to the triangle. This will be reflected in the rectangle with the argument $initUpper - 1$. The rectangle is bounded by the lines $x = i_o$, $x = i_f$, $y = initLower$, $y = initUpper - 1$.

In procedural terms, we return
sumTriangleB(i, i_o, i_f, j, c, d, f, false, false) +
sumRectangle(i, i_o, i_f, j, initLower, initUpper - 1, f)

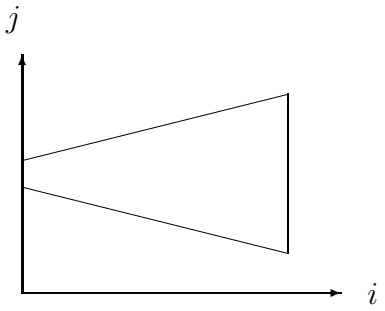
3.4.4 $A < 0$ and $C = 0$



The next case is when $a < 0$ and $c = 0$. By the assumptions on the boundary lines, the two boundary lines cannot intersect. The area enclosed is an upper rectangle and a triangle below it with horizontal upper base and vertical line at the right. The rectangle is bounded by the lines $x = i_o$, $x = i_f$, $y = initUpper$, $y = initLower + 1$. Again the $initLower + 1$ is due to giving a shared boundary line to the triangle.

In procedural terms, we return
sumTriangleB(i, i_o, i_f, j, a, b, f, true, false) +
sumRectangle(i,i_o, i_f, j, initLower + 1, initUpper)

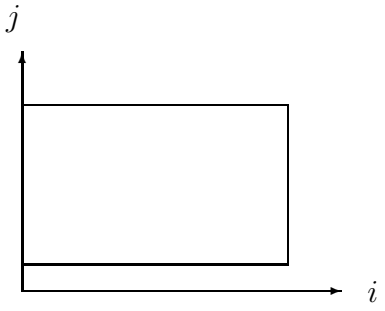
3.4.5 $A < 0$ and $C > 0$



The next case is when $a < 0$ and $c > 0$. By the assumption on the boundary lines, the boundary lines cannot intersect. The area enclosed is two triangles, with a rectangle between the two triangles, with all three figures possibly sharing a common horizontal line. The first triangle lies above the rectangle with a horizontal lower base at the line $y = initUpper$ and vertical line at the right The second triangle is below the rectangle with a horizontal upper base at the line $y = initLower$ and vertical line to the right. The rectangle is bounded by the lines $x = i_o$, $x = i_f$, $y = initLower + 1$, $y = initUpper - 1$. Again the 1 corresponds to the triangle receiving the overlapping area. We note that if all three figures share a common horizontal line, the rectangle will subtract the area which is double counted by the two triangles, due to a MAPLE behavior on summations. We return the upper triangle, the lower triangle and the rectangle.

In procedural terms, we return
sumTriangleB(i, i_o, i_f, j, c, d, f, false, false) +
sumTriangleB(i, i_o, i_f, j, a, b, f, true, false) +
sumRectangle(i, i_o, i_f, j, initLower + 1, initUpper - 1)

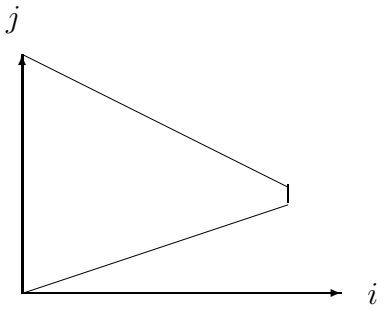
3.4.6 $A = 0$ and $C = 0$



The next case is when $a = 0$ and $c = 0$, which corresponds to a rectangular region. This region is bounded by the lines $x = i_o$, $x = i_f$, $y = b$, $y = d$. We remark that $initLower = finalLower = b$ and $initUpper = finalUpper = d$. In this case we return this rectangle.

In procedural terms, we return
sumRectangle(i, i_o, i_f, j, initLower , initUpper)

3.4.7 $A > 0$ and $C < 0$



The next case is when $a > 0$ and $c < 0$. This case corresponds to an upper triangle with horizontal lower base at the line $y = finalUpper$ and vertical line to the left, a lower triangle with horizontal upper base at the line $y = finalLower$ and vertical line to the left, and a rectangle in between the two triangles. The potential overlap of the two boundary lines would correspond to the two lines meeting at the point $(i_f, finalLower) = (i_f, finalUpper)$. In other words, the two triangles formed by the boundary lines may or may not meet in a point.

Intersecting boundary lines

If it can be determined that the two triangles meet at a point, $(a \cdot i_f + b = c \cdot i_f + d)$, then the horizontal line $y =$ *finalUpper*, which includes this point, represents an area which will be computed by both triangles. As a result, we need to subtract the area of the rectangle bounded by the lines $x = i_o$, $x = i_f$, $y =$ *finalLower*, $y =$ *finalUpper*. Here, we note that *finalLower* = *finalUpper*.

In procedural terms, we return
sumTriangleB(i, i_o, i_f, j, c, d, f, false, true) +
sumTriangleB(i, i_o, i_f, j, a, b, f, true, true) -
sumRectangle(i, i_o, i_f, j, finalLower, finalUpper)

Possible non intersecting boundary lines

If we cannot determine if the two triangles meet at a point, then we cannot determine the difference. Define

$$theDiff = finalUpper - finalLower$$

If *theDiff* were to equal 0, we would have the sub case defined above, where we subtracted a rectangle. If *theDiff*, was greater than 0 then we would want to add the area of the rectangle bounded by the horizontal lines, $y = finalLower + 1$ and $y = finalUpper - 1$, (possibly zero area) to the summation. Since we have two different results depending upon a difference which cannot be determined, we rely on the MAPLE behavior for summations.

The sum from $\alpha + k$ to α where $k > 2$ is the negative of the sum from $\alpha + 1$ to $\alpha + (k - 1)$ Therefore, in either case, we can simply return the two triangles and the rectangle with appropriate bounds. In other words, if the two lines meet, the behavior for the program is the same. If the two triangles share a boundary line, the rectangle

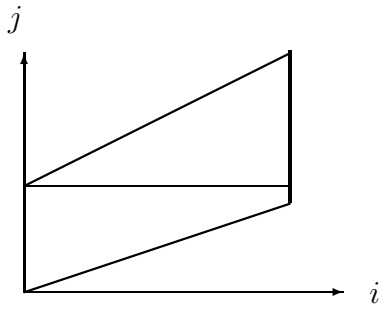
will subtract this area. If they do not, the rectangle will add the appropriate area. We return the upper triangle, the lower triangle and the rectangle.

In procedural terms, we return
sumTriangleB(i, i_o, i_f, j, c, d, f, false, true) +
sumTriangleB(i, i_o, i_f, j, a, b, f, true, true) +
sumRectangle(i, i_o, i_f, j, finalLower + 1, finalUpper - 1)

3.4.8 $A > 0$ and $C > 0$

The next case is when $a > 0$ and $c > 0$. This case can correspond to different areas depending upon if the lower boundary line, $a \cdot i + b$ intersects the horizontal line containing the initial upper boundary line point, $c \cdot i_o + d$. We will go through these two cases and describe why additional information is needed on this possible intersection.

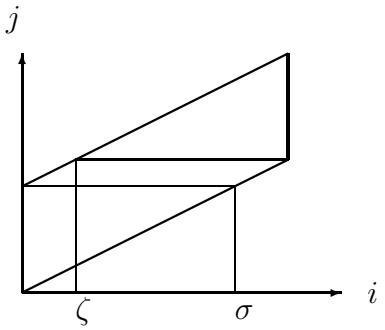
No intersection of lower boundary line with $y = initialUpper$



If the lower boundary line $a \cdot i + b$ is strictly less than or equal to any value (y-value) the upper boundary line achieves, the area is simply a rectangle and two triangles. Specifically, there is an upper triangle with horizontal lower base at the line $y = initialUpper$ and vertical line on the right. There is a lower triangle with horizontal upper base at the line $y = finalLower$ and vertical line on the left. There is also a rectangle of possibly zero area (MAPLE sum zero) bounded by the lines $x = i_o$, $x = i_f$, $y = finalLower + 1$, $y = initialUpper - 1$. The area of the rectangle will not be subtracted by the assumption that the largest value *finalLower* can achieve is *initialUpper*. The rectangle is not guranteed to add area either.

In procedural terms, we return
sumTriangleB(i, i_o, i_f, j, c, d, f, false, false) +
sumTriangleB(i, i_o, i_f, j, a, b, f, true, true) +
sumRectangle(i, i_o, i_f, j, finalLower + 1, initialUpper - 1)

Intersection of $y = a \cdot i + b$ with $y = initialUpper$



If the lower boundary line $a \cdot i + b$ is strictly greater (for all values of i large enough) than the line $y = initUpper$, the area enclosed can include a trapezoid. The point of interesction between the horizontal line containing the initial upper boundary line point, $y = initialUpper$ and the lower boundary line needs to be determined in terms of i. In other words, the value of i when $initialUpper = a \cdot i + b$ needs to be determined.

We represent this intersection point by σ in the above diagram and formally define **initIntersect**.

$$initIntersect := (initialUpper - b)/a$$

If $initIntersect < outerFinal$ then there is either a non right angle triangular area or a trapezoidal area with a triangular area. The existence of these regions depend on if the lower boundary line intersects the upper boundary line when $i = i_f$.

No intersection of boundary lines intersect in final y-values

If the lower boundary line does not intersect the upper boundary line at i_f (shown in above diagram), then there is a difference between the points corresponding to $c \cdot i_f + d$ and $a \cdot i_f + b$, which corresponds to an upper triangular area with a right angle. More precisely, there is an upper triangular region with lower horizontal base at the line $y = finalLower$, and a vertical line on the right. The starting value of i for this triangle needs to be determined. The starting value is equal to i when $c \cdot i + d = finalLower$.

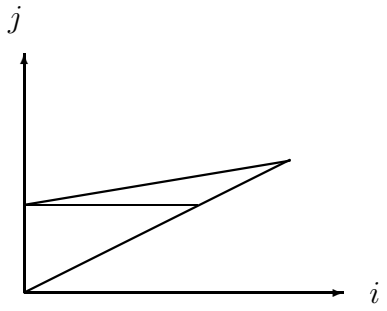
We show this point in the above diagram by ζ and now formally define **triIntersect**.

$$triIntersect := \frac{(finalLower - d)}{c}$$

There is also a trapezoidal area bounded by the lines $y = initialUpper$, $y = finalLower$, $y = c \cdot i + d$, $y = a \cdot i + b$. We return the area of the upper triangle, the area of the trapezoid, plus the area of the lower triangle.

In procedural terms, we return
sumTriangleB(i, triIntersect, i_f, j, c, d, f, false, false) +
'sum'('sum'(f, i=$\lceil (j-d)/c \rceil$..$\lfloor (j-b)/a \rfloor$), j =initUpper+1..finalLower-1)+
sumTriangleB(i, i_o, initIntersect, j, a, b, f, true, true)

Intersection of boundary lines at final y-values



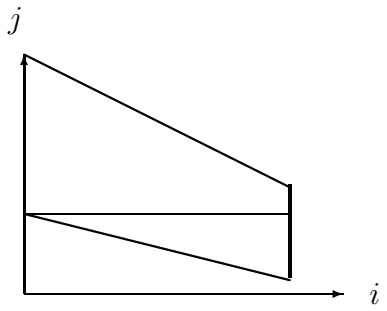
If the two lines intersect at this point $(i_f, \mathit{finalUpper}) = (i_f, \mathit{finalLower})$, then there is a triangular area without a right angle (above $y = \mathit{initUpper}$). The boundary lines of this triangle are $y = \mathit{initialUpper}$, $y = c \cdot i + d$, $y = a \cdot i + b$. For this area we return a double sum, remembering not to double count the area of the lower triangle. We also need to return the lower triangle, which we do by passing possible a non-integer value to sumTriangleB. This is valid since the denominator of the value will be equal to the slope of the triangle. Therefore we return the upper triangle and the lower triangle.

In procedural terms, we return
'sum'('sum'(f, i =$\lceil (j-d)/c \rceil$..$\lfloor (j-b)/a \rfloor$), j=initUpper + 1..finalUpper) +
sumTriangleB(i, i_o, initIntersect, j, a, b, f, true, true)

3.4.9 $A < 0$ and $C < 0$

The next case is when $a < 0$ and $c < 0$. This case can correspond to different areas depending upon if the upper boundary line, $c \cdot i + d$ intersects the horizontal line containing the initial lower boundary line point, $a \cdot \mathit{outerInitial} + b$. We will go through these two cases and describe why additional information is needed on this possible intersection.

No intersection of upper boundary line with $y = \mathit{initLower}$

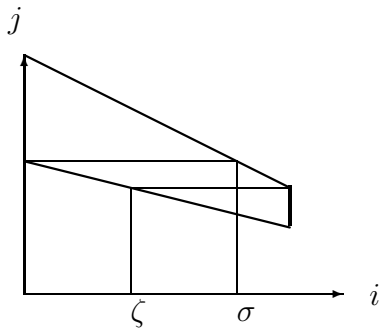


If the the upper boundary line $c \cdot i + d$ is strictly greater than or equal to the line $y = \mathit{initLower}$, the area is simply rectangles and triangles. Specifically, there is an upper triangle with horizontal lower base at the line $y = c \cdot i_f + d$ and vertical line on the left. There is a lower triangle with horizontal upper base at the line $y = a \cdot i_o + b$ and vertical line on the right. There is also a rectangle of possibly zero area (MAPLE sum zero) bounded by the lines $x = i_o$, $x = i_f$, $y = \mathit{initialLower} + 1$, $y = \mathit{finalUpper} - 1$. In

this case, the area of the rectangle will never be subtracted as its its summation variable will be from low to high.

In procedural terms, we return
sumTriangleB(i, i_o, i_f, j, c, d, f, false, true) +
sumTriangleB(i, i_o, i_f, j, a, b, f, true, false) +
sumRectangle(i, i_o, i_f, j, initLower + 1, finalUpper - 1)

Intersection of upper boundary line with with $y = initialLower$



If the upper boundary line $c \cdot i + d$ is strictly less (for all values of i large enough) than the value of the lower boundary line at $i = i_o$, defined as $y = initialLower$, the area enclosed can include a trapezoid. The point of intersection between $y = initialLower$ and the upper boundary line needs to be determined in terms of i. In other words, the value of i when $initialLower = c \cdot i + d$ needs to be determined. We show this intersection point in the diagram above with σ and now formally define **initIntersect**.

$$initIntersect := (initialLower - d)/c$$

We now look at where this intersection takes place, in terms of i. If $initIntersect < i_f$, then below the horizontal line, $y = initialLower$ there is either a non right angle triangular area or a trapezoidal area with a triangular area. The existence of these regions depend on if the lower boundary line intersects the upper boundary line at $i = i_f$.

No intersection of boundary lines in final y-values

If the upper boundary line does not intersect the lower boundary line at $i = i_f$ (shown in above diagram), then there is a difference between the points $(i_f, c \cdot i_f + d)$ and $(i_f, a \cdot i_f + b)$. This difference defines a lower triangular area with a right angle. More precisely, there is an lower triangular region with upper horizontal base at the line $y = finalUpper$ (shared boundaries given to triangles), and a vertical line on the right. The starting value of i for this triangle needs to be determined. The starting value is equal to i when $a \cdot i + b = finalUpper$. We show this intersection in the above diagram with ζ and now formally define **triIntersect**.

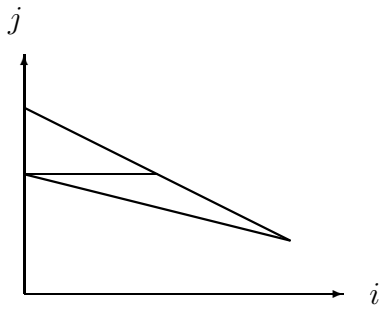
$$triIntersect := \frac{(finalUpper - b)}{a}$$

There is also a trapezoidal area bounded by the lines $y = initialLower - 1$, $y = finalUpper + 1$, $y = c \cdot i + d$, $y = a \cdot i + b$. We return the area of the triangle plus the

area of the trapezoid when $triIntersect = \frac{(finalUpper - b)}{a}$. In this case we return the upper triangular area, the lower triangular area and the trapezoidal area.

In procedural terms, we return
sumTriangleB(i, i_o, initIntersect, j, c, d, f, false, true)+
sumTriangleB(i, triIntersect, i_f, j, a, b, f, true, false) +
'sum'('sum'(f, i =$\lceil \frac{(j-b)}{a} \rceil$..$\lfloor \frac{(j-d)}{c} \rfloor$), j =initLower-1..finalUpper+1)

Intersection of boundary lines at final y-values



If the two boundary lines intersect at this point $((i_f, finalUpper) = (i_f, finalLower))$, there is a triangular area without a right angle. The boundary lines of this triangle are $y = initialLower\text{-}1$, $y = c \cdot i + d$, $y = a * i + b$. For this area we return the area of the original upper triangle and a double sum for this triangle area.

In procedural terms, we return
sumTriangleB(i, i_o, initIntersect, j, c, d, f, false, true) +
'sum'('sum'(f, i =$\lceil \frac{(j-b)}{a} \rceil$..$\lfloor \frac{(j-d)}{c} \rfloor$), j =initialLower-1..finalLower)

Chapter 4

Simplifying Harmonic Summations

In this chapter we will use the procedure to aid in the finding of closed forms for particular summations involving Harmonic numbers. The particular case will be to show the desired sum and identify it as one of the nine cases. Then we will evaluate the sum using MAPLE, and then once again after interchanging the order of summation. In this section we will often write the Harmonic numbers as an explicit sum $\sum_{j=1}^{k} \frac{1}{j}$ instead of H_k. We have that $H_n \approx \ln(n) + \gamma$ where $\gamma = 0.57721\cdots$ is Euler's constant which means asymptotically the two functions are similar.

MAPLE uses the Psi function instead of the harmonic numbers to represent the sum of the reciprocals of the first n numbers. Therefore, this section will contain the Psi function. If the MAPLE procedure returned a result involving the Psi function, the result, if possible, would be switched to the harmonic number representation using the identity $H_n = \Psi(n+1) - \gamma$.

4.1 Example harmonic summations

In this section, all bounds of the inner summation will be of the form $a = 0, c = 1$ unless otherwise stated. This area corresponds in most cases to a triangle.

4.1.1 $\sum_{k=1}^{n} H_k$

Here we will sum the Harmonic numbers.

$$\sum_{k=1}^{n} H_k \tag{4.1}$$

We first write this as the double summation,

$$\sum_{k=1}^{n} \sum_{j=1}^{k} \frac{1}{j} \tag{4.2}$$

MAPLE gives the following closed form for this expression, when written as H_k. The reason for this behavior could be an internal table which MAPLE references the summation.

$$H_{n+1}(n+1) - n - 1 \tag{4.3}$$

MAPLE gives the following closed form for this expression when H_k is written explicitly. We change the Ψ function into the harmonic function when possible to simplify the expressions.

$$((n+1)^2 - n - 4)H_{n+1} + (5 - (n+1)^2 + 2n)H_{n+2} - 3 \tag{4.4}$$

The result of interchanging the order of summation is

$$\sum_{j=1}^{n}\sum_{k=j}^{n}\frac{1}{j} = (n+1)H_n - n \tag{4.5}$$

We note that if we were to write this in terms of H_{n+1} then the $\frac{1}{n+1}$ term would be multiplied by $\frac{1}{n+1}$ and would equate to 1. Hence we would need to subtract 1 from the sum and this equivalent form would be equation 4.3.

4.1.2 $\sum_{k=1}^{2n}(-1)^k H_k$

Here we will look at summing the alternating Harmonic numbers.

$$\sum_{k=1}^{2n}(-1)^k H_k \tag{4.6}$$

We note that this series is telescoping. We write the above summation as

$$\sum_{k=1}^{2n}\sum_{j=1}^{k}(-1)^k\frac{1}{j} \tag{4.7}$$

MAPLE does not give a closed form for this expression, when written as H_k. MAPLE simply returns equation 4.6 as labeled.

MAPLE gives the following closed form for this expression when H_k is written out explicitly.

$$\frac{-1}{2}\gamma((-1)^{2n+1}+1) + \sum_{k=1}^{2n}(-1)^k\Psi(k+1) \tag{4.8}$$

We note that 4.8 is simply writing harmonic(k) as $\Psi(k+1)+\gamma$ and then extracting the γ terms from the summation.

The result of interchanging the order of summation is

$$\sum_{j=1}^{2n}\sum_{k=j}^{2n}\frac{(-1)^k}{j} = -\frac{1}{2}(-1)^{2n+1}H_{2n} + \frac{1}{4}(H_n - \Psi(n+\frac{1}{2})) - \frac{1}{2}\ln(2) \tag{4.9}$$

Summation 4.7 is actually a Telescoping series. MAPLE does not seem to look for simple telescoping series although that is the method that it uses to sum rational functions. We may write sum 4.7 as $-H_1+H_2-\ldots-H_{2n-1}+H_{2n}$. The terms can be grouped together and we may also rewrite this summation as $(H_2-H_1)+(H_4-H_3)+\ldots+(H_{2n}-H_{2n-1}) = \frac{1}{2}H_n$

4.1.3 $\sum_{k=1}^{n} kH_k$

Here we will look at summing k times each kth Harmonic number.

$$\sum_{k=1}^{n} kH_k \tag{4.10}$$

We write the above summation as

$$\sum_{k=1}^{n}\sum_{j=1}^{k} k\frac{1}{j} \tag{4.11}$$

MAPLE gives the following closed form expression, when written as H_k.

$$\frac{1}{2}H_{n+1}(n+1)^2 - \frac{1}{4}(n+1)^2 + \frac{3}{4}(n+1) - \frac{1}{2}(n+1)(H_{n+1}+1) \tag{4.12}$$

MAPLE gives the following closed form for this expression when H_k is written out explicitly.

$$(\frac{1}{4}(n+1)^2 + \frac{1}{2}n - \frac{1}{4})(n+1)H_{n+1} + \frac{1}{4}(n+1-(n+1)^2(n+2)H_{n+2} \tag{4.13}$$

The result of interchanging the order of summation is

$$\sum_{j=1}^{n}\sum_{k=j}^{n}\frac{k}{j} = -\frac{1}{4}(n+1)^2 + \frac{3}{4}n + \frac{1}{4} + \frac{1}{2}n(n+1)H_n \tag{4.14}$$

We note that both equation 4.13 and 4.14 simplify to

$$\frac{1}{2}nH_{n-1}(n+1) - \frac{1}{4}n^2 + \frac{3}{4}n + \frac{1}{2} = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4} \tag{4.15}$$

4.1.4 $\sum_{k=1}^{n}(-1)^k kH_k$

Here we will look at summing the alternating series of k times each kth Harmonic number. We write the above summation as

$$\sum_{k=1}^{n}\sum_{j=1}^{k}(-1)^k k\frac{1}{j} \tag{4.16}$$

MAPLE returns equation 4.16 when written as H_k.

MAPLE gives the following closed form for this expression when H_k is written out explicitly.

The result of interchanging the order of summation is

$$\begin{gathered} -\tfrac{1}{2}(-1)^{n+1}H_n\Big(n+\tfrac{1}{2}-\tfrac{1}{4}((-1)^{n+1}+1\Big) \\ +\tfrac{1}{8}\Big(2\ln(2)+(-1)^{n+1}\Psi(1+\tfrac{1}{2}n)+(-1)^n\Psi(\tfrac{1}{2}(n+1))\Big) \end{gathered}$$

We note that again we are finding the half integer Ψ function. We note this is because different results are to be returned if n is an odd integer or if n is an even integer.

4.1.5 $\sum_{k=1}^{n} k^2 H_k$

Here we will look at summing k^2 times each kth Harmonic number.

$$\sum_{k=1}^{n} k^2 H_k \tag{4.17}$$

We write the above summation as

$$\sum_{k=1}^{n} \sum_{j=1}^{k} k^2 \frac{1}{j} \tag{4.18}$$

MAPLE gives the following closed form expression, when written as H_k.

$$(n+1)^3\Big(\frac{1}{3}H_{n+1} - \frac{1}{9}\Big) + (n+1)^2\Big(-\frac{1}{2}H_{n+1} + \frac{1}{12}\Big) + (n+1)\Big(\frac{1}{6}H_{n+1} + \frac{1}{36}\Big) \tag{4.19}$$

MAPLE gives the following closed form for this expression when H_k is written out explicitly.

$$n^3\Big(\frac{1}{3}H_{n-1} - \frac{1}{9}\Big) + n^2\Big(\frac{1}{2}H_{n-1} + \frac{5}{12}\Big) + n\Big(\frac{1}{6}H_{n-1} + \frac{19}{36}\Big) + \frac{1}{6} \tag{4.20}$$

The result of interchanging the order of summation is

$$-(n+1)^3\frac{1}{9} + (n+1)^2\frac{5}{12} - n\frac{17}{36} - \frac{11}{36} + \frac{1}{6}n(2n^2+3n+1)H_n \tag{4.21}$$

Equation 4.21 simplifies to equation 4.20.

4.1.6 $\sum_{k=1}^{n} (-1)^k k^2 H_k$

Here we will look at summing the alternating series of k^2 times each $k$thHarmonic number.

$$\sum_{k=1}^{n} (-1)^k k^2 H_k \tag{4.22}$$

We write the above summation as

$$\sum_{k=1}^{n} (-1)^k k^2 H_k \tag{4.23}$$

MAPLE gives the following closed form expression, when written as H_k.

$$-\tfrac{1}{8}\tfrac{1}{(n+1)^2}\Big(6(n+1)^2 - 3n - 4\Big)\Big((-1)^{n+1}(n+1)^2 H_{n+1}\Big) - \tfrac{1}{8}\tfrac{1}{n+2}$$
$$\Big(2n+1\Big)\Big((-1)^{n+2}(n+2)^2 H_{n+2}\Big) + \tfrac{1}{8}$$

MAPLE gives the following closed form for this expression when H_k is written out explicitly.

$$\frac{1}{2}(-1)^n n^2 H_{n-1} + \frac{3}{4}n(-1)^n + \frac{1}{2}(-1)^n H_{n-1} + \frac{1}{8} + \frac{3}{8}(-1)^n \tag{4.24}$$

The result of interchanging the order of summation is the same as equation 4.24 when simplified.

4.1.7 $\sum_{k=1}^{n} \frac{1}{k} H_k$

Here we will look at summing $\frac{1}{k}$ times each kth Harmonic number.

$$\sum_{k=1}^{n} \frac{1}{k} H_k \tag{4.25}$$

We write the above summation as

$$\sum_{k=1}^{n} \sum_{j=1}^{k} \frac{1}{k} \frac{1}{j} \tag{4.26}$$

MAPLE does not simplify equation 4.25 when written as H_k, when written out explicitly or after interchanging the order of summation. After interchanging the order of summation it is apparent that H_n^2 is in the solution. This is shown in the equation below. The interchanged form is

$$\sum_{j=1}^{n} \sum_{k=j}^{n} \frac{1}{k} \frac{1}{j} = \sum_{j=1}^{n} \frac{1}{k} \sum_{k=j}^{n} \frac{1}{j} = \sum_{j=1}^{n} \frac{1}{j} (H_n - H_{j-1}) \tag{4.27}$$

The closed form is $\frac{1}{2}\Big(H_n^2 + H_n^{(2)}\Big)$.

4.1.8 $\sum_{k=1}^{n} \frac{H_k}{k+1}$

Here we will look at summing $\frac{1}{k+1}$ times each kth Harmonic number.

$$\sum_{k=1}^{n} \frac{H_k}{k+1} \tag{4.28}$$

We write the above summation as

$$\sum_{k=1}^{n} \sum_{j=1}^{k} \frac{1}{k+1} \frac{1}{j} \tag{4.29}$$

MAPLE gives the following closed form expression, when written as H_k.

$$\frac{1}{2}\Big(H_{n+1}^2 + \Psi(1, n+2) - \frac{\pi^2}{6} - (1-\gamma)^2 - 2\gamma + 1\Big) \tag{4.30}$$

MAPLE does not give a closed form when H_k is written out explicitly or after interchanging the order of summation. We note that equation 4.29 gives the closed form

$$\frac{1}{2}\Big(H_{n+1}^2 - H_{n+1}^{(2)}\Big) \tag{4.31}$$

Chapter 5

Test Cases

In this section we will test the procedures described previously. We will use simple functions to allow for MAPLE to find a closed form expression. We are only proving the correctness of the procedures in this section, and are not attempting to show its effectiveness.

5.1 Decide Case

The general format of this section will be to show a summation, MAPLE's evaluation of that expression, which may have the simplify procedure applied to it. Then the same summation after the order has been interchanged will be shown. This will also be evaluated for a closed form, and possibly simplified. Then, the difference of the two summations (original and interchanged) will be displayed. If the result is not zero, as often occurs if the new summation involves floor or ceiling functions, then numerical values for n will be substituted and the resultant summations numerically evaluated, with the difference being displayed.

5.2 $A < 0$ and $C = 0$

The MAPLE display to the input *''sum(sum(* $i + j, j = -i + 2..1), i = 1..n$*)'* is

$$\sum_{i=1}^{n} \sum_{j=-i+2..1}^{1} i + j \tag{5.1}$$

We then evaluate the summation which gives us the left hand side and after simplification gives us the right hand side of below.

$$\frac{1}{2}(n+1)^2 - \frac{2}{3}n - \frac{2}{3} + \frac{1}{6}(n+1)^3 = \frac{1}{6}n^3 + n^2 + \frac{5}{6}n$$

After calling *sumOrderChange* (*''sum(sum(* $i + j, j = -i + 2..1), i = 1..n$*)'*) we see that this particular summation only involves a triangular area, which can be seen by inspecting the bounds of the inner summation when i is at its minimum value, 1. We have that

$-i+2 = -(1)+2 = 1$ and hence the lower and upper boundary lines intersect at this initial value of i. The result of this procedure call is

$$\sum_{j=-n+2}^{1} \sum_{i=-j+2}^{n} (i+j) \tag{5.2}$$

When simplified this expression gives the same value as 5.1. The result of evaluation is

$$n^2 + \frac{4}{3} + \frac{5}{6}n - \frac{1}{2}(-n+2)n^2 - \frac{1}{2}n(-n+2)^2 - \frac{1}{6}(-n+2)^3$$

5.3 $A = 0$ and $C > 0$

This example consists of a triangular region which sits above a rectangular region. The triangular region is upper bounded by the line $y = -1$. Notice how this line is given to the triangle function and not to the rectangle function in 5.4. The MAPLE display to the input *''sum(sum($i^2 j, j = -i..1), i = 1..n$)'* is

$$\sum_{i=1}^{n} \sum_{j=-i}^{1} i^2 j \tag{5.3}$$

When evaluated this summation returns

$$\frac{5}{12}(n+1)^3 - \frac{5}{8}(n+1)^2 + \frac{11}{60}(n+1) - \frac{1}{10}(n+1)^5 + \frac{1}{8}(n+1)^4$$

After calling *sumOrderChange* (*''sum(sum($i^2 j, j = -i..1), i = 1..n$)'*), we are returned

$$\sum_{j=-n}^{-1} \sum_{i=-j}^{n} i^2 j + \sum_{j=0}^{1} i^2 j \tag{5.4}$$

This is evaluated into two results, which are

$$\Big(-\frac{1}{10}n^5 - \frac{3}{8}n^4 - \frac{1}{12}n^3 + \frac{3}{8}n^2 + \frac{11}{60}n \Big) + \Big(\frac{1}{3}(n+1)^3 - \frac{1}{2}(n+1)^2 + \frac{1}{6}(n+1) \Big)$$

The evaluations of 5.3 and 5.1 agree when they simplified.

5.4 $A < 0$ and $C > 0$

We now present an example in which we have two triangle regions which are separated by a rectangular region. The common boundary line between the rectangle and one of the triangles will be given to the respective triangle which can be seen in5.6. The MAPLE display to the input *''sum(sum($i + j, j = -i..i), i = 1..n$)'* is

$$\sum_{i=1}^{n} \sum_{j=-i}^{i} i + j \tag{5.5}$$

The simplified evaluation to 5.5 is

$$\frac{2}{3}n^3 + \frac{3}{2}n^2 + \frac{5}{6}n$$

After calling *sumOrderChange* $('\text{'}sum(sum(\ i+j, j=-i..i), i=1..n)')$, we are returned

$$\sum_{j=1}^{m}\sum_{i=j}^{n}(i+j) + \sum_{j=-n}^{-1}\sum_{i=-j}^{n}(i+j) + \sum j = 0^0 \sum_{i=1}^{n}(i+j) \tag{5.6}$$

This equation simplifies to the same result as 5.5.

5.5 $A = 0$ and $C = 0$

When both a and c are 0, we have just a rectangular region. This first example will be a single line. The MAPLE display to the input $'\text{'}sum(sum(\ i+j, j=1..1), i=1..n)'$ is

$$\sum_{i=1}^{n}\sum_{j=1}^{1} i+j \tag{5.7}$$

This equation evaluates to the left hand side and simplifies to the right hand side of

$$\frac{1}{2}(n+1)^2 + \frac{1}{2}n - \frac{1}{2} = \frac{1}{2}n^2 + \frac{3}{2}n$$

After interchanging the order of summation by calling
sumOrderChange $('\text{'}sum(sum(\ i+j, j=1..1), i=1..n)')$, we are returned

$$\sum_{j=1}^{1}\sum_{i=1}^{n} i+j \tag{5.8}$$

This evaluates to and simplifies to the exactly as 5.7 and hence the two summations are equal.

We now present a slightly less trivial example in which a rectangle is summed over as opposed to just a line. The MAPLE display to the input $'\text{'}sum(sum(\ i+j, j=1..k), i=1..n)'$ is

$$\sum_{i=1}^{n}\sum_{j=1}^{k} i+j \tag{5.9}$$

This equation evaluates to the left hand side and simplifies to the right hand side of the following.

$$\frac{1}{2}(n+1)k^2 + \frac{1}{2}k(n+1)^2 - \frac{1}{2}(k^2+k) = \frac{1}{2}k^2n + \frac{1}{2}kn^2 + kn$$

After interchanging the order of summation by calling
sumOrderChange $('\text{'}sum(sum(\ i+j, j=1..k), i=1..n)')$, we are returned

$$\sum_{j=1}^{k}\sum_{i=1}^{n} i+j \tag{5.10}$$

This evaluates to the same as 5.9 with n and k interchanged, and simplifies to the same as 5.9

5.6 $A > 0$ and $C < 0$

With the conditions on a and c we are looking at two triangles which may meet each other or may share a common boundary line. The MAPLE behavior on sums allows us to give a single return expression when interchanging the order of summation. We will go through three examples where the difference between the final two points is 0, 1 and 2.

In this first test, the difference between the final two boundary lines is 0 and hence they intersect at their final values. The MAPLE display to the input *''sum(sum(* $i+j, j = i-2\cdot n.. -i), i = 1..n)'$ is

$$\sum_{i=1}^{n}\sum_{j=i-2n}^{-i} i+j \tag{5.11}$$

This equation gives a long expression which simplifies to

$$\frac{1}{2}(n)^2+\frac{1}{6}n-\frac{2}{3}n^3$$

After interchanging the order of summation by calling *sumOrderChange* $(''sum(sum(\ i+j, j = i-2\cdot n.. -i), i = 1..n)')$, we are returned two triangles with the area of the common boundary line subtracted through a rectangular area which is degenerated into just a line. The returned summation is

$$\sum_{j=-n}^{-1}\sum_{i=1}^{-j}(i+j)+\sum_{j=1-2n}^{-n}\sum_{i=1}^{j+2n}(i+j)-\sum_{j=-n}^{-n}\sum_{i=1}^{n}(i+j) \tag{5.12}$$

This evaluates to an even longer expression than 5.11 and simplifies to the same as 5.11 and hence the two summations are equal.

5.7 $A = 0$ and $C < 0$

Here we will look at a triangle region with a possible empty rectangular region below it. This case will not have the two boundary lines touching which is seen by looking at the inner summation bounds, specifically $n+1-i = n+1-n = 1 \neq 0$. The MAPLE display to the input *''sum(sum(* $\frac{1}{i+j}, j = 0..n+1-i), i = 1..n)'$ is

$$\sum_{i=1}^{n}\sum_{j=0}^{n+1-i} i+j \tag{5.13}$$

This equation returns an expression involving the Ψ function. The evaluation of 5.13 is

$$\Big((n+1)^2-n-2)(\Psi(n+2)-\Psi(n+1)\Big)+1$$

After interchanging the order of summation by calling *sumOrderChange* $(''sum(sum(\ \frac{1}{i+j}, j = 0..n+1-i), i = 1..n)')$, we are returned a triangle and a rectangle, with the shared boundary line at $y = 1$ given to the triangle. The returned summation is

$$\sum_{j=1}^{n}\sum_{i=1}^{-j+n+1}\frac{1}{i+j}+\sum_{j=0}^{0}\sum_{i=1}^{n}\frac{1}{i+j} \tag{5.14}$$

This evaluates to

$$\Big(5-(n+1)^2+2n\Big)\Big(\Psi(n+2)-\Psi(n+3)\Big)-\Psi(n+2)+3+\Psi(n+1)$$

Both of these returned forms simplify to $\frac{(n+2)n}{n+1}$ and hence are equal.

5.8 $A > 0$ and $C = 0$

Here we look at a triangular region with a possible empty rectangular region above it. In this first test, the two boundary lines do not touch at the final value. The MAPLE display to the input *''sum(sum($i+j, j=i+2..n+4), i=1..n$)'* is

$$\sum_{i=1}^{n}\sum_{j=i+2}^{n+4} i+j \tag{5.15}$$

This equation returns a polynomial expression which is

$$\frac{1}{2}n^3+4n(n+1_+\frac{7}{2}n-1+\frac{1}{2}n(n+1)^2+\frac{3}{2}(n+1)^2-\frac{1}{2}(n+1)^3$$

After interchanging the order of summation by calling
sumOrderChange (*''sum(sum(* $i+j, j=i+2..n+4), i=1..n$*)'*), we are returned a triangle and a rectangle, with the shared boundary line at $y=n+2$ given to the triangle. The returned summation is

$$\sum_{j=3}^{n+2}\sum_{i=1}^{j-2}(i+j)+\sum_{j=n+3}^{n+4}\sum_{i=1}^{n}(i+j) \tag{5.16}$$

This evaluates to

$$\frac{1}{2}(n+3)^3-\frac{5}{2}(n+3)^2+1+(n+1)^2+(n+1)(n+3)+(n+1)(n+4)$$

The difference of these two summations is 0 and hence they are equal.

The following test has the two boundary lines touch in the final value. The MAPLE display to the input *''sum(sum(* $\frac{1}{j(j+2)}, j=i..n), i=1..n$*)'* is

$$\sum_{i=1}^{n}\sum_{j=i}^{n}\frac{1}{j(j+2)} \tag{5.17}$$

This equation returns a polynomial which is a complicated expression involving a rational function of n multiplied by the Ψ function. After interchanging the order of summation by calling
sumOrderChange (*''sum(sum(* $\frac{1}{j(j+2)}, j=i..n), i=1..n$*)'*), we are returned a triangle. The returned summation is

$$\sum_{j=1}^{n}\sum_{i=1}^{j}\frac{1}{j(j+2)} \tag{5.18}$$

This evaluates to

$$\Psi(n+3)-\frac{3}{2}+\gamma$$

This closed form agrees with MAPLE simplification of 5.17 when values for n are inputted, with $n \leq 1000$.

Chapter 6

Quicksort

6.1 The Quicksort Algorithm

Quicksort is a recursive sorting algorithm which sorts a list of elements by selecting a pivot element (uniformly at random) from the list of elements and then partitioning the other elements into two sublists, those which are less than the pivot and those which are greater than the pivot. The algorithm then recursively sorts the two sublists. When there is only one element in each list, the lists are put back together to form a sorted list.

We let n be the number of elements which are to be sorted. The worst case running time of this algorithm corresponds to selecting either the minimum or maximum element in the list at each stage of the algorithm, requiring $n-1$ pivots to be selected and $\sum_{i=n-2}^{1} i = \sum_{i=1}^{n-2} i = \frac{(n-2)(n-1)}{2}$ comparisons to be performed. However, this is far from the expected running time or average case running time of Quicksort which is known to be $n\log(n)$. In this chapter, we will analyze the expected number of comparisons of the Quicksort algorithm (the average run time) as well as provide some insight into how to calculate the variance of the algorithm. The methods used will give an exact solution, which asymptotically agrees with the big O $n\log(n)$ running time.

We will use the terminology of referring to the elements by their position in the sorted list, after the algorithm is done. We use position i to refer to the i^{th} position of the sorted array. We have that position i in the sorted will correspond to position i' of the input array with i' not necessarily equal to i.

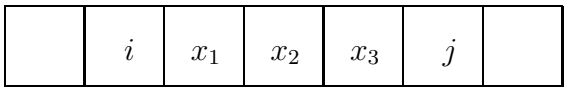
6.2 Expected Value of Quicksort

To compute the expected value of the total number of comparisons in the Quicksort algorithm, we will use indicator random variables. We first let X be a random variable which denotes the number of comparisons in the Quicksort algorithm. We let X_{ij} be a $0,1$ indicator random variable which is 1 if position i is compared with position j, and 0 otherwise.

Probability two elements compared

Assuming unique elements of the input array, the probability that two positions $i, j : (j > i)$ will be compared is proportional to the distance between them.

We look at the following diagram to illustrate this point.



$$Pr[i, j \text{ compared}] = \frac{Pr[i]+Pr[j]}{Pr[i]+Pr[x_1]+Pr[x_2]+Pr[x_3]+Pr[j]}$$

By assuming the algorithm does not bias in selecting which pivot element to use, we have that the probability that position i will be compared to position j is just the probability that i or j will be selected before any element which is between them in the sorted array. Therefore we have

$$Pr[X_{ij} = 1] = \frac{2}{j - i + 1} \tag{6.1}$$

Expected number of comparisons

We now analyze the expected number of comparisons. Since X_{ij} is an indicator random variable, we have 6.2.

$$X = \sum_{1 \le i < j \le n} X_{ij} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij} \tag{6.2}$$

We now take the expected value of X.

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} Pr[X_{ij} = 1] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j - i + 1} \tag{6.3}$$

MAPLE does not give a closed form for 6.3. We therefore interchange the order of summation and are returned an equivalent summation.

$$sumOrderChange('sum(sum(\frac{2}{j - i + 1}, j = i + 1..n), i = 1..n - 1)');$$

$$\sum_{j=2}^{n} \sum_{i=1}^{j-1} \frac{2}{j - i + 1} \tag{6.4}$$

MAPLE gives a closed form for 6.4 which when simplified is

$$2n\Psi(n) - 4n + 2n\gamma + 2\gamma + 2 + 2\Psi(n) + \frac{1}{n}$$

This is in fact equal to $2(n+1)H_n - 4n$. Therefore we have that the expected number of comparisons of the Quicksort algorithm is $2(n + 1)H_n - 4n$ which is roughly $2n \log(n)$, which is quite fast.

6.3 Variance of Quicksort

In this section we will calculate part of the variance of the Quicksort algorithm. To find the variance, we calculate the second moment $E[X^2]$.

$$E[X^2] = \sum_{i=1}^{n}\sum_{j=i+1}^{n} E[X_{ij}^2] + 2\sum_{i=1}^{n}\sum_{j=i+1}^{n}\sum_{r=1}^{n}\sum_{s=r+1}^{n} E[X_{ij}X_{rs}] = E[X] + 2B$$

The four way sum B is broken into cases depending upon the ordering or i, j, r, s. We will look at the case when $(i < j < r < s)$. We will refer to this summation as $B1_a$ and it is given in 6.5

Subcase $B1_a$

$$B1_a := \sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{2}{j-i+1} \sum_{r=j+1}^{n}\sum_{s=r+1}^{n} \frac{2}{s-r+1} \tag{6.5}$$

We now look at the innermost summation and we use an change of variables. We take the denominator $s-r+1$ and call it k. When $s = r+1$, we have $k = (r+1)-r+1 = 2$. When $s = n$ we have $k = n-r+1$. We can now write the innermost summation as $H_{n-r+1} - 1$, with a coefficient of 2. We therefore have (pulling out the constants)

$$B1_a := 4\sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{1}{j-i+1} \sum_{r=j+1}^{n} \left(H_{n-r+1} - 1 \right) \tag{6.6}$$

Breaking on the minus sign of 6.6

We now break this summation on the minus sign and we first sum the latter term, the -1. Of this new expression, the innermost summation is just the number of terms in the summand and we can reduce this to 6.7

$$B1_{aS} := -4\sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{1}{j-i+1} \sum_{r=j+1}^{n} 1 = -4\sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{1}{j-i+1}\Big(n-j\Big) \tag{6.7}$$

Summing the n term of 6.7

Again we separate 6.7 and we now treat one the summation where the numerator does not contain j. We pull out the n term and arrive at 6.8.

$$B1_{aS1} := -4n\sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{1}{j-i+1} \tag{6.8}$$

The innermost summation is the same procedure as before. We let $k = j-i+1$, the denominator. When $j = i+1$, $k = (i+1)-i+1 = 2$. When $j = n$, $k = (n)-i+1 = n-i+1$. We then have

$$B1_{aS1} = -4n\sum_{i=1}^{n}\sum_{k=2}^{n-i+1} \frac{1}{k} = -4n\sum_{i=1}^{n}\sum_{k=1}^{n-i+1} \frac{1}{k} + 4n\sum_{i=1}^{n} 1 \tag{6.9}$$

The $\frac{1}{k}$ term corresponds to H_{n-i+1}. The latter summation equates to the number of terms being summed over, which is $n+1$. In order to sum H_{n-i+1} with respect to i, we write this as the difference of two summations. $H_{n-i+1} = H_n - H_i$. We therefore arrive at 6.10.

$$B1_{aS1} = -4n\sum_{i=1}^{n} H_n - 4n\sum_{i=1}^{n} H_i + 4n(n+1) \tag{6.10}$$

The first term is summing over H_n which does not depend on i and hence the actual summation evaluates to $(n+1)H_n$. The second summation is summing the harmonic numbers which evaluate to $(n+1)(H_{n+1}-1)$. Therefore we have the following.

$$B1_{aS1} = 4n\bigg(n+1-(n+1)H_n-(n+1)(H_{n+1}-1)\bigg) \tag{6.11}$$

Summing the j term of 6.7

We now treat the other part of the summation in 6.7.

$$B1_{aS2} := 4\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{j}{j-i+1} \tag{6.12}$$

We set k to the denominator, and then $j = k+i-1$. The bounds of summation are from 2 to $n-i+1$. We therefore write 6.12 as

$$B1_{aS2} = 4\sum_{i=1}^{n}\sum_{k=2}^{n-i+1}\frac{k+i-1}{k} = 4\sum_{i=1}^{n}\bigg(\Big((n-i+1)-1\Big)+\Big(\sum_{k=1}^{n-i+1}\frac{i-1}{k}\Big)-(i-1)\bigg) \tag{6.13}$$

The last $(i-1)$ is since we start k at 1 in the innermost summation and so we need to subtract this added value. This process results in a nicer form to the solution. The $n-i$ term is from the $\frac{k}{k}$ term of the fraction. We collect $n-i$ and $1-i$ to get $n-2i+1$, and then look at the double summation. We write the inner summation as $(i-1)H_{n-i+1} = (i-1)\Big(H_n - H_i\Big)$. Therefore 6.13 becomes

$$B1_{aS2} = 4\sum_{i=1}^{n}(n-2i+1) + 4\sum_{i=1}^{n}(i-1)H_n - 4\sum_{i=1}^{n}(i-1)H_i$$

$$B1_{aS2} = 4\sum_{i=1}^{n}(n-2i+1) + 4H_n\sum_{i=1}^{n}(i-1) - 4\Big(\sum_{i=1}^{n} iH_i - \sum_{i=1}^{n} H_i\Big)$$

We have that $\sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}$. Therefore, this summation becomes straight forward.

6.3.1 Summing over H_{n-r+1} term of 6.6

We now return to equation 6.6 to sum over the H_{n-r+1} term, which is invariably more difficult. We recall this summation as 6.14.

$$B1_{aT} := 4\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{j-i+1}\sum_{r=j+1}^{n} H_{n-r+1} \tag{6.14}$$

In our usual tradition, we break the innermost summation and write it as the difference of two harmonics, $H_n - H_r$. We note that we could have also have used a change of variables for r which would sum H_k from $k = 1$ to $k = n - j$.

$$B1_{aT} = 4\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{j-i+1}(n-j)H_n - 4\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{j-i+1}\sum_{r=j+1}^{n}H_r$$

The first summation is not of interest to us since this is the same as 6.8 multiplied by H_n. We then look at the second summation which we can write as

$$B1_{aTs} = 4\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{j-i+1}\sum_{r=j+1}^{n}H_r$$

We have that

$$\sum_{r=j+1}^{n}H_r = \sum_{k=1}^{n}H_k - \sum_{k=1}^{j}H_k = (n+1)(H_{n+1}-1) - (j+1)(H_{j+1}-1)$$

We remark that we have done summations of this type except for the term corresponding $(j+1)H_{j+1}$. We will now look at 6.14 restricted to summing over this term. We will even pull out the 4 from the summation and only care about the form of the answer. Specifically we will be interested in 6.15.

$$B1_{aM} := \sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{j-i+1}(j+1)H_{j+1} \tag{6.15}$$

Interchanging the order of summation

We now interchange the order of summation. We have that 6.15 is equivalent to the following.

$$b1_{aMS} = \sum_{j=2}^{n}\sum_{i=1}^{j-1}\frac{1}{j-i+1}(j+1)H_{j+1}$$

Since we are summing over i, we wish to make the denominator appear in terms of $i - \alpha$. This is to allow us to write the innermost summation as the difference of two Harmonics. We let $\alpha = (j+1)$.

$$b1_{aMS} = \sum_{j=2}^{n}\sum_{i=1}^{j-1}\frac{-1}{i-(j+1)}(j+1)H_{j+1}$$

We now again change the index of summation and use a new variable k.

$$\sum_{j=2}^{n}\sum_{k=2}^{j}\frac{1}{k}(j+1)H_{j+1} = \sum_{j=2}^{n}\left(H_j(j+1)H_{j+1} - (j+1)H_{j+1}\right) \tag{6.16}$$

We will first evaluate the $(j+1)H_{j+1}$ term (the second term) and then manipulate the first term and sum the form which was not done previously. The second term is of the

form kH_k. We know that $\sum_{k=1}^{n} kH_k = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}$. Therefore we simply apply this form when letting $k = j + 1$.

$$\sum_{j=2}^{n}(j+1)H_{j+1} = \sum_{k=3}^{n+1} kH_k = \sum_{k=1}^{n+1} kH_k - \sum_{k=1}^{2} kH_k$$

We now look at the first term of 6.16, $H_j(j+1)H_{j+1}$, which we write as $H_{j+1}(j+1)H_{j+1} - H_{j+1}$. Since we have already completed summations corresponding to $-H_{j+1}$ we will only sum over the first term.

$$\sum_{j=2}^{n}(j+1)H_{j+1}^2 = \sum_{k=1}^{n-1} kH_k^2$$

Since $\sum_{k=1}^{n} kH_k^2 = \frac{n(n-1)}{2}H_n^2 + \frac{1-n^2+n}{2}H_n + \frac{n(n-3)}{4}$, we simply replace n by $n-1$ and we are done.

Concluding Quicksort Remarks

Therefore we see that all parts of our original summation 6.5 can be solved for. The above was to give a demonstration of the necessary manipulations needed to perform such summations involving the harmonic numbers. The methods used for finding the closed form expressions for summations involving the harmonic numbers are mostly solved using an interchange of the order of summation. Although in the finding a closed form for 6.5 we only performed one such interchange, in the known closed form summations which we used, the interchange was implicit.

Chapter 7

Symbolic Summation in MAPLE

Following [1], for definite summations in which the bounds are known and are constant in value, the MAPLE function *add* is used, as MAPLE does not need to find a closed form expression for the summation. For finite sums, *add* is equivalent to *sum*

$$\sum_{i=1}^{4} i = \text{add}(i, i = 1..4);$$

The evaluation of both sides of the equality of the above expression will evaluate to 10.

Another type of summation is one in which the range is infinite. MAPLE first represents the summation in closed form (if possible) and then computes the value of the limit. The following shows the results of MAPLE calls for finding a closed form for $\sum_{i=1}^{n} \frac{1}{i^2}$ and then taking the limit as n goes to infinity. This is the same as if we set $n = \infty$ in the initial MAPLE call.

$$sum(\frac{1}{i^2}, i = 1..n) = -\Psi(1, n+1) + \frac{1}{6}\pi^2$$

$$\text{limit}(-\Psi(1, n+1) + \frac{1}{6}\pi^2, n = \text{infinity}) = \frac{1}{6}\pi^2$$

$$sum(\frac{1}{x^2}, x = 1..\infty) = \frac{1}{6}\pi^2$$

The following is due to [1]. Given a summation involving a_k, MAPLE looks for an expression s_k not involving a summation sign such that $a_k = s_k - s_{k-1}$. The following methods are used to find s_k. Summing polynomials, (the summation of k^i over i) are done using Bernoulli polynomials. If s_k is a hypergeometric term, $\frac{s_k}{s_{k-1}} \in Q(k)$, then Gosper's decision procedure is used. The extended Gosper algorithm is used if $a_k = s_k - s_{k-m}, m > 1$

7.1 Rational Functions

We will now elaborate upon the method used to sum rational functions. A rational function of x is the fraction of two polynomials with coefficients from $F[x]$. In formula, $f(x) \in F(x)$ is a rational function if $f(x) := \frac{p(x)}{q(x)} : p(x), q(x) \in F[x]$. Moreover we add

the restriction that $gcd(p(x), q(x)) = 1$. We are now interested in summations such as $\sum_{i=1}^{n} f(x)$. To sum rational function, MAPLE attempts to compute rational functions $h(x)$ and $r(x)$ such that $f(x) = h(x+1) - h(x) + r(x)$. We remark that $r(x)$ is attempted to be kept to the smallest denominator degree possible, as shown in [3] We now turn our attention to $q(x)$ and its shift equivalent classes. We first factor $q(x)$ and then group factors in the same class if they are shift equivalent. Two factors $q_i(x), q_j(x)$ are shift equivalent if $q_i(x) = q_j(x+m) : m \in Z$. We call the multiplicity of $q_i(x)$ the number of times $q_i(x)$ appears in $q(x)$. The shift classes admit irreducible factors of multiplicity greater than 1.

Example shift class

Consider $f(x) := \frac{1}{x^3+x^2}$. We have $q(x) = x^3 + x^2$. The irreducible factors of $q(x)$ are $q_1(x) := x$ which has multiplicity 2 and $q_2(x) := (x+1)$ which has multiplicity 1. We have that $q_1(x)$ and $q_2(x)$ are in the same shift class since in the above definition, with $m = 1$.

Now, consider $f(x) := \frac{1}{x^2+\frac{x}{2}}$. We have $q(x) = x^2 + \frac{x}{2}$. The irreducible factors of $q(x)$ are $q_1(x) := x$ and $q_2(x) := (x + \frac{1}{2})$. We have that $q_1(x)$ and $q_2(x)$ are not in the same shift class since in the above definition, since $m = \frac{1}{2}$ is not an integer.

Dispersion

Let the maximum m for a shift equivalent class be called the dispersion of that shift class. Let the dispersion of $f(x)$ be the maximum among all shift equivalent classes. From [3] we have that if the sum has a rational closed form, it has non-zero dispersion. As a direct consequence we have the Harmonic numbers which have dispersion 0 do not admit a rational closed form. Although we can write them in terms of other harmonic functions or the Psi function, we cannot write closed forms for summations involving harmonic numbers which are the ratio of two polynomials. A slow way to calculate the dispersion is to find the greatest integer d such that $\gcd(q(x), q(x+d)) \neq 1$, i.e. they have a non trivial common divisor.

7.1.1 Moenck's algorithm

Moenck's algorithm takes in $f(x)$ and finds all shift equivalent classes. Within each shift equivalent class, we first fill in all integers between the two factors which give the maximal dispersion. For example if the shift equivalent class contained x of multiplicity 2 and $(x+3)$ of multiplicity 1, we would add the factors $(x+1)$ and $(x+2)$. We then take each of these factors to the highest multiplicity of that class. The shift structure of our example class would then be $x^2(x+1)^2(x+2)^2(x+3)^2$.

We then use partial fraction decomposition on these "filled in" shift classes. We note that by construction all shift classes are relatively prime. As pointed out in [4] we can find a closed form for $f(x)$ by considering each one of these shift classes separately. We iteratively repeat the procedure until we have remainder terms with dispersion 0 or remainder terms that are 0. We then sum the pieces. As pointed out in [3], Moenck's method does not always work in MAPLE because it does not make sure the shift classes are relatively prime. An improved algorithm is due to Paule [2].

We will now provide a simple example of Moenck's algorithm.

$$\sum_{x=1}^{n} \frac{1}{x(x+2)} \tag{7.1}$$

We will now look at $f(x) := \frac{1}{x(x+2)}$. There is only one shift equivalent class and so we write $f(x)$ as

$$\frac{x+1}{x(x+1)(x+2)} := \frac{\alpha(x)}{\beta(x)} = \frac{\gamma(x+1)}{\delta(x+1)} - \frac{\gamma(x)}{\delta(x)} + \frac{\epsilon(x)}{\nu(x)}$$

By [4], we set

$$\delta(x) = \gcd(\beta(x), \beta(x-1)) = \gcd\Big(x(x+1)(x+2), (x-1)(x)(x+1)\Big) = x(x+1)$$

$$\nu(x) = \frac{\beta(x)}{\delta(x)} = \frac{x(x+1)(x+2)}{x(x+1)} = x+2$$

An S-form for a rational function has a saturated monic denominator and the gcd of the numerator and denominator is 1. By Paule, if $r, s \in K(x)$ both proper with S-forms $\langle \alpha, \beta \rangle$ and $\langle \gamma, \delta \rangle$, then we have that $\deg(\gamma) = \deg(\alpha) - \deg(\beta) + \deg(\delta) + 1$ which implies the degree of γ is $1 - 3 + 2 + 1 = 1$. Although we do not meet the requirements for this proposition, we use this estimate for the degree of γ. Therefore we can write γ as $Ax + b$, and we estimate $\epsilon(x)$ as x. Solving this system of linear equations, we find

$$\gamma(x) = -x - \frac{1}{2}$$

Therefore $f(x) = \frac{\gamma(x+1)}{\delta(x+1)} - \frac{\gamma(x)}{\delta(x)}$ where $\gamma(x)$ and $\delta(x)$ are defined as before. For completeness, we repeat these assignments.

$$\delta(x) = x(x+1)$$

$$\gamma(x) = -x - \frac{1}{2}$$

Returning to our summation, we can write

$$\sum_{k=1}^{n} \frac{1}{x(x+2)} = \sum_{k=1}^{n} \Big(\frac{\gamma(k+1)}{\delta(k+1)} - \frac{\gamma(k)}{\delta(k)} \Big) = \frac{\gamma(n+1)}{\delta(n+1)} - \frac{\gamma(1)}{\delta(1)}$$

Writing this as a closed form, we have equation 7.1 is equal to

$$\frac{3}{4} - \frac{n + \frac{3}{2}}{n^2 + 3n + 2} \tag{7.2}$$

We note that for $n = 1$ both the sum and the closed form evaluate to $\frac{1}{3}$. For $n = 2$, both equate to $\frac{11}{24}$.

MAPLE returns the same result as 7.2.

$$sum(\frac{1}{x \cdot (x+2)}, x = 1..n) = -\frac{1}{2}\frac{3 + 2n}{(n+2)(n+1)} + \frac{3}{4}$$

Chapter 8

Conclusion

Using a symbolic language such as MAPLE to simplify and find closed form expressions for summations which arise out of algorithms is beneficial, although not completely self reliant. User manipulation is still needed to derive the desired form of the solution. Examples have been shown where the MAPLE procedure *sum* returned a closed form involving the Psi function, although it returns a form involving the harmonic numbers in other cases. The user must manipulate one expression into the other to arrive at a consistent solution form.

Furthermore, not all summations can be calculated by MAPLE, and a change of variables is often needed to arrive at the expected closed form. This method could be automated, although manipulations of this sort seem to be more application specific. A common change of variables in the analysis of Quicksort was for the denominator of $\frac{1}{j-i+1}$. In a more general setting, a manipulation of this sort may not lead to simpler expressions.

Overall, the programming language MAPLE seems to be best at finding closed forms for functions which are polynomials in i and j. MAPLE only simplified basic summations involving the harmonic numbers or their equivalent forms, and the more involved summations used user manipulations in conjunction with the simpler identities MAPLE proved.

Future work should either become domain specific and automate many of the manipulations shown in the analysis of the Quicksort algorithm, or it should become very broad and look into other summation techniques. The interchanging of the order of summation seems to have limited potential. If either the lower or the upper boundary line is not a constant, strictly increasing or strictly decreasing function, we do not expect MAPLE to find simple closed forms. This is since the area which is being summed over will have to be broken into summations, with each one working over a restricted range of the area. Moreover, if the boundary lines are not easily invertible, there is little chance for MAPLE to find a closed form. This was shown by MAPLE not finding closed forms for expressions which involved the floor and ceiling functions, although their equivalent summations when the order of summation was interchanged gave closed forms. Overall, we conclude that although there exists limitations in the use of interchanging the order of summation, there are examples which prove it is a positive addition to a symbolic programming language summation tools.

Bibliography

[1] A. Heck Introduction to Maple. Springer-Verlag, New York, 1993

[2] P. Paule Greatest Factorial Factorization and Symbolic Summation. *J. Symbolic Computation* **11** (1995)

[3] R. Piratsu Algorithms for Indefinite Summation of Rational Functions in Maple. *The Maple Technical Newsletter* **2** (1995)

Appendix A

MAPLE Source Code

A.1 Parsing the input

```
sumOrderChange := proc (doubSum)
# Matthew Dailey, February 2009, Worcester Polytechnic Institute
# ASSUMPTIONS
# the outer lower bound is a number
# the outer upper bound will probably be n, but could be a number
# a,b,c,d coefficients of the inner sum will be numbers or
    variables
description "Interchange the order of summation for a double
    summation with constant or linear bounds":
local input, output, innerSum, innerRange, outerRange, innerLow,
    innerHigh, innerVar, outerLow,outerHigh, outerVar,
    funct,linearParser,sumToReturn, a,b,c,d:
input := doubSum:
# ======================================================
# Parse the sum

if op(0,input) <> sum then
    return "invalid arguments":
end if:

#the full sum(f(i,j),j=1..n)
innerSum := op(1,input):

# if we have a double sum, then the first operand should be sum
if op(0,innerSum)<>sum then
    return "invalid arguments":
end if:

#process this sum
funct := op(1, innerSum):
innerRange := op(2, innerSum):
```

```
#inner range here means j=1..10  parses to =, j, 1..10
innerVar := op(1,innerRange):

#innerRange is range(min,max)
innerLow := op(1, op(2,innerRange)):
innerHigh := op(2,op(2 ,innerRange)):

#parses into range, 1, 10
outerRange := op(2,input):
outerVar := op(1,outerRange):

#outerRange is range (min,max)
outerLow := op(1, op(2,outerRange)):
outerHigh := op(2,op(2 ,outerRange)):

#=======================================================

# assume that a and c are integers
# innerLow and innerHigh are the values that will give us a,b,c,d
# a*var + b
# c*var + d

if type(innerLow, linear(outerVar)) then
    b:= subs(outerVar=0, innerLow):
    a:= (innerLow-b)/(outerVar):
else
    a:=0:
    b:=innerLow:
end if:

if type(innerHigh, linear(outerVar)) then
    d:= subs(outerVar=0, innerHigh):
    c:= (innerHigh-d)/(outerVar):
else
    c:=0:
    d:=innerHigh:
end if:

# if the case is forced then pass this information
if nargs > 2 then
  return decideCase(innerVar, outerVar, outerLow, outerHigh, a, b,
    c, d, funct, ops(3,args) ):
else
  return decideCase(innerVar, outerVar, outerLow, outerHigh, a, b,
    c, d, funct):
```

```
end if:

end proc:
```

A.2 Decide Case

```
with(difforms):    # for type const
# copyright Matthew Dailey February 2009 Worcester Polytechnic
    Institute

# Procedure to input information regarding a double summation and
    return the appropriate summation with the order of summation
    interchanged
# The outer variable is outerVar, and the inner variable is
    innerVar.
# The initial outerVar value is outerInitial.
# The final outerVar value is outerFinal
# innerVar ranges from a*outerVar+b (lower boundary line) to
    c*outerVar+d (upper boundary line)
# Assumption: the range is from low to high
# funct is a function of outerVar and innerVar
decideCase:=proc(innerVar, outerVar, outerInitial, outerFinal, a,
    b, c, d, funct)

description "Interchange the order of summation":

local initUpper, initLower, finalUpper, finalLower, theDiff,
    outerMax, initialLineDiff, initIntersect,
    triIntersect,theInitDiff:
# initUpper is the initial y value of the upper boundary line
# initLower is the initial y value of the initial boundary line
# finalUpper is the final y value of the upper boundary line
# finalLower is the final y value of the lower boundary line
# theDiff is the difference between the boundary lines at
    outerFinal
# initialLineDiff is the difference between the boundary lines at
    outerInitial
# initIntersect is often used to correspond to outerVar when one
    boundary line intersects the horizontal line y = init_X X =
    initLower|initUpper
# triIntersect is for the value of outerVar for a specific case
    described when appropriate

initLower:= a*outerInitial+b:
finalLower:= a*outerFinal + b:
initUpper:= c*outerInitial + d:
finalUpper := c*outerFinal + d:

# see if the two boundary lines start at the same point
if type(initLower, constant) and type(initUpper, constant) then
```

```
# Check to see if the summation is possible
if initUpper < initLower and (initLower - initUpper) > 1 then
  print("The assumption is the inner summation bounds go from
  low to high");
  # Return the original expression
  return 'sum'( 'sum'( funct,innerVar =
  a*outerVar+b..c*outerVar+d),
  outerVar=outerInitial..outerFinal);
end if:

# The two boundary lines start at the same point, results in
  simpler cases
if initLower = initUpper then
  if a = 0 and c = 0 then
    # just a line
    return 'sum'( subs(innerVar=initLower, funct), outerVar =
  outerMin..outerMax);
    #return 'sum'('sum'(funct, outerVar = outerMin..outerMax),
  innerVar = initLower..initLower):
  elif a = 0 then
    # we have a triangle with horizontal line at the bottom
  (i.e. at b)
    return sumTriangleB(outerVar, outerInitial, outerFinal,
  innerVar, c, d, funct,false, false):
  elif c = 0 then
    # we have a triangle with horizontal line at the top (i.e.
  at d)
    return sumTriangleB(outerVar, outerInitial, outerFinal,
  innerVar, a, b, funct,true, false):
  elif (c>0) and (a< 0) then
    # we have a sideways v, do both triangles minus the double
  counted line
    return sumTriangleB(outerVar, outerInitial, outerFinal,
  innerVar, c, d, funct,false, false) +
      sumTriangleB(outerVar, outerInitial, outerFinal, innerVar,
  a, b, funct,true, false) -
      'sum'('sum'(funct, outerVar = outerMin..outerMax),
  innerVar = initLower..initLower):
  elif a>0 and c<0 then
    # This is a contradiction to our assumption unless it is an
  empty sum
    if (type(outerMax, constant) = false) or (outMin <>
  outerMax) then
      printf("Violation of assumptions on bounds of innerSum\n");
    end if:
```

```
    end if:
  end if:
end if:

# It cannot be determined if the two boundary lines start at the
    same point.
# We will have 9 cases depending on a and c
# theInitDiff is the difference between the y values of the
    starting points of the two boundary lines
theInitDiff := initUpper - initLower;

# theDiff is the difference between the y values of the ending
    points of the two boundary lines
theDiff := finalUpper - finalLower;


if (a = 0 and c < 0) then
  if type( theDiff, constant) then    # if it is a number

    if theDiff = 0 then
      # The area is a triangle, the two boundary lines do not
    intersect
      return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, c, d, funct, false, true);

    elif theDiff > 0 then
      # The area is a triangle on top of a rectangle
      return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, c, d, funct, false, true) +
        sumRectangle( outerVar, outerInitial, outerFinal,
    innerVar, initLower, finalUpper - 1, funct);

    elif theDiff < 0 then
      # This is a violation on the assumptions, however a sum will
    attempted to be returned
      # The area will be a triangle, however it will not range
    over all values of outerVar
      outerMax :=  (b-d)/c :
      if c <> 1 and c <> (-1) then
        outerMax := floor( outerMax ):
      end if:
      if theDiff < (-1) then
        printf("Violation on assumption on boundary lines at final
    y-values, attempted solution of where assumptions hold
    returned\n");
        return sumTriangleB( outerVar, outerInitial, outerMax,
```

```
    innerVar, c, d, funct, false, true):
      elif theDiff = -1 then
        # we simply exclude the values of OuterVar after the two
    lines cross
        return sumTriangleB( outerVar, outerInitial, outerMax,
    innerVar, c, d, funct, false, true):
      end if:
    end if:
  end if:

  # It cannot be determined if the two areas intersect
  # return a triangle and a rectangle area.
  return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, c, d, funct, false, true) +
    sumRectangle( outerVar, outerInitial, outerFinal, innerVar,
    initLower, finalUpper - 1, funct);

elif a > 0 and c = 0 then
  if type( theDiff, constant) then    # if it is a number

    if theDiff = 0 then
      # The area is a triangle, the two boundary lines do not
    intersect
      return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, a, b, funct, true, true);

    elif theDiff < 0 then
      # there is a triangle below a rectangle
      return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, a, b, funct, true, true) +
        sumRectangle( outerVar, outerInitial, outerFinal,
    innerVar, finalLower + 1, finalUpper, funct);

    elif theDiff > 0 then
      # This is a violation on the assumptions, however a sum will
    attempted to be returned
      # The area will be a triangle, however it will not range
    over all values of outerVar
      outerMax := (d-b) / a:
      if a <> 1 and a <> -1 then
        outerMax := floor( outerMax):
      end if:
      if theDiff < -1 then
        printf("Violation on assumption on boundary lines at final
    y-values, attempted solution of where assumptions hold
    returned\n");
```

```
        sumTriangleB( outerVar, outerInitial, outerMax, innerVar,
    a, b, funct, true, true):
      elif theDiff = -1 then
        # we simply exclude the values of OuterVar after the two
    lines cross
        sumTriangleB( outerVar, outerInitial, outerMax, innerVar,
    a, b, funct, true, true):
      end if:
    end if:
  end if:

  # It cannot be determined if the two areas intersect
  # return a triangle and a rectangle area.
  return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, a, b, funct, true, true) +
    sumRectangle( outerVar, outerInitial, outerFinal, innerVar,
    finalLower + 1, finalUpper, funct):

elif  a = 0 and c > 0 then
  if type(theInitDiff, constant) and theInitDiff = 0 then
    # only a triangle
    return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, c, d, funct, false, false);
  else
    # A triangle with a rectangle below it
    return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, c, d, funct, false, false) +
      sumRectangle( outerVar, outerInitial, outerFinal, innerVar,
    initLower, initUpper - 1, funct):
  end if:

elif  a < 0 and c = 0 then
  if type(theInitDiff, constant) and theInitDiff = 0 then
    # only a triangle
    return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, a, b, funct, true, false);
  else
    # A triangle with a rectangle above it
    return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, a, b, funct, true, false) +
      sumRectangle( outerVar, outerInitial, outerFinal, innerVar,
    initLower + 1, initUpper, funct):
  end if:

elif  a < 0 and c > 0 then
  # two triangles with a rectangle between them
```

```
  # works the same if or if not there is a rectangle between the
    triangles
  return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, c, d, funct, false, false) +
    sumTriangleB( outerVar, outerInitial, outerFinal, innerVar, a,
    b, funct, true, false) +
    sumRectangle( outerVar, outerInitial, outerFinal, innerVar,
    initLower + 1, initUpper - 1, funct):

elif  a = 0 and c = 0 then
  # A rectangular region
  # if the two boundary lines will return an empty summation
  if type(theInitDiff, constant) and (initUpper - 1) = initLower
    then
    return 0:
  end if:
  return sumRectangle( outerVar, outerInitial, outerFinal,
    innerVar, initLower, initUpper, funct):

elif  a > 0 and c < 0 then
  # two triangles with a rectangle between them.
  # if the two boundary lines meet in a point, we can subtract the
    area of the line computed by both triangles
  if theDiff = 0 then
    return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, c, d, funct, false, true) +
      sumTriangleB( outerVar, outerInitial, outerFinal, innerVar,
    a, b, funct, true, true) -
      sumRectangle( outerVar, outerInitial, outerFinal, innerVar,
    finalUpper, finalUpper, funct):
  end if:
  # Maple will handle adding missing area or subtracting for a
    repeated line (boundary of both triangles)
    return sumTriangleB( outerVar, outerInitial, outerFinal,
    innerVar, c, d, funct, false, true) +
      sumTriangleB( outerVar, outerInitial, outerFinal, innerVar,
    a, b, funct, true, true) +
      sumRectangle( outerVar, outerInitial, outerFinal, innerVar,
    finalLower + 1, finalUpper - 1, funct):

elif  a > 0 and c > 0 then
  # simplified greatly if two lines start in the same place
  # we do not include the integer height line between lower non
    right triangle and upper right triangle
  if type(theInitDiff, constant) and theInitDiff = 0 then
    return sum(sum( funct, i=ceil( (j-d)/c )..floor( (j-b)/a )),
```

```
  j=initLower..finalLower-1)+
    sumRectangle( outerVar, (a*outerFinal+b-d)/c, outerFinal,
  innerVar, c, d, funct, false, false);
end if:

# This case heavily depends on if the lower boundary line
  intersects the horizontal line y = initUpper
# and also if the lower line surpasses the initial upper
  horizontal line
# moreover, we care if the final y values of the two boundary
  lines are equivalent
initialLineDiff := initialUpper - finalLower:
if type( initialLineDiff, constant) then
  if initialLineDiff >= 0 then
    # No intersection
    if initialLineDiff = 1 then
      # do not have a rectangle between the two triangles
      return sumTriangleB(outerVar, outerInitial, outerFinal,
  innerVar, c, d, funct, false, false) +
        sumTriangleB(outerVar, outerInitial, outerFinal,
  innerVar, a, b, funct, true, true):
    end if:
    return sumTriangleB(outerVar, outerInitial, outerFinal,
  innerVar, c, d, funct, false, false) +
      sumTriangleB(outerVar, outerInitial, outerFinal, innerVar,
  a, b, funct, true, true) +
      sumRectangle( outerVar, outerInitial, outerFinall,
  innerVar, finalLower + 1, initUpper - 1, funct):
  end if:

  # There is an intersection
  initIntersect := (initUpper - b) / a:  # outerVar value at
  which the intersection occurs

  # Need to determine if the two boundary points intersect at
  final value
  if type( theDiff, constant) then
    if theDiff = 0 then
      # intersection at final value
      return sumTriangle(outerVar, outerInitial, initIntersect,
  innerVar, a, b, funct, true, true) +
        sum( sum( funct, outerVar = ceil( (innerVar - d)/c
  )..floor( (innerVar - b)/a )), innerVar = initUpper +
  1..finalUpper):
    elif theDiff > 0 then
      # The value of outerVar on the upperboundary line when it
```

```
    achieves the value finalLower
        # this value marks the start of a triangluar region
        triIntersect := (finalLower - d) / c:

        # return this upper triangle, the lower triangle, and a
    double sum for area in between
        return sumTriangleB( outerVar, triIntersect, outerFinal,
    innerVar, c, d, funct, false, false) +
          sumTriangle(outerVar, outerInitial, initIntersect,
    innerVar, a, b, funct, true, true) +
          'sum'( 'sum'( funct, outerVar = ceil( (innerVar - d)/c
    )..floor( (innerVar - b)/a )), innerVar = initUpper +
    1..finalUpper - 1):
      else
        # Violation the assumption on final y- values
        printf("Violation on boundary lines for case a>0 and
    c>0\n"):
        return 0:
      end if:
    end if:

    # Need user input to determine if the final y values of the
    two boundary lines are the same
    # if they are then we return the lower triangle + the double
    sum for area above y = initUpper
    # else we need to return the lower triangle, an upper triangle
    + a double sum for the area between.
    # For now, the below will work but is messy
    return sumTriangle(outerVar, outerInitial, initIntersect,
    innerVar, a, b, funct, true, true) +
      'sum'( 'sum'( funct, outerVar = ceil( (innerVar - d)/c
    )..min( outerFinal, floor( (innerVar - b)/a ))), innerVar =
    initUpper + 1..finalUpper):
  end if:

  # it cannot be determined if the lower boundary line ever
    achieves a y value of initUpper
  # We will need user input
  printf("Need information on whether the lower boundary line
    achives value %d, returning 0\n", initUpper):

elif  a < 0 and c < 0 then
  # simplified greatly if two lines start in the same place
  # we do not include the integer height line between upper non
    right triangle and lower right triangle
  if type(theInitDiff, constant) and theInitDiff = 0 then
```

```
    return 'sum'('sum'( funct, i=ceil( (j-b)/a )..floor( (j-d)/c
    )), j=initLower..finalUpper-1)+
      sumTriangleB( outerVar, (c*outerFinal+d-b)/a, outerFinal,
    innerVar, a, b, funct, true, false);
  end if:

  # This case heavily depends on if the upper boundary line
    intersects the horizontal line y = initLower
  # and also if the two boundary lines intersect in a point at
    outerVar = outerFinal
  #initLower,finalLower,initUpper,finalUpper
  if type(finalUpper-initLower,constant) and finalUpper >=
    initLower then
    return sumTriangleB(outerVar, outerInitial, outerFinal,
    innerVar, c,d,funct,false,true)+
      sumTriangleB(outerVar, outerInitial, outerFinal, innerVar,
    a,b,funct,true,false)+
      sumRectangle(outerVar, outerInitial, outerFinal, innerVar,
    initLower+1,finalUpper-1,funct);
  else
    # define where the top boundary lines intersects initialLower
    initIntersect := (initLower - d)/c;
    theDiff := finalUpper - finalLower;
    if type(theDiff, constant) then
      if theDiff <> 0 then
        triIntersect := (finalUpper-b)/a;
        return sumTriangleB(outerVar, outerInitial, initIntersect,
    innerVar, c,d,funct,false,true)+
          sumTriangleB(outerVar, triIntersect, outerFinal,
    innerVar, a,b,funct,true,false)+
          'sum'('sum'(funct, outerVar = ceil( (innerVar-b)/a
    )..floor( (innerVar-d)/c ),
    innerVar=initLower-1..finalUpper+1));
      else
        return sumTriangleB(outerVar, outerInitial, initIntersect,
    innerVar, c, d, funct, false,true)+
          'sum'('sum'(funct, outerVar = ceil( (innerVar-b)/a
    )..floor( (innerVar-d)/c ) ),innerVar =
    initialLower-1..finalLower);
      end if:
    end if:
  end if:
end if:

# we do not have enough information to interchange the order of
    summation
```

```
# return the original summation
return 'sum'( 'sum'( funct,innerVar = a*outerVar+b..c*outerVar+d),
    outerVar=outerInitial..outerFinal);

end proc:
```

A.3 Sum Rectangle

```
sumRectangle := proc(outerVar, outerInitial, outerFinal, innerVar,
    innerLower, innerUpper, funct)

# first check if this will be an empty sum, meaning the outer
    summation goes from v to v-1.
if (innerLower - innerUpper) = 1  then
  return 0:
end if:

# returns a double summation of the rectangle with the order of
    summation reversed
return 'sum'('sum'(funct,outerVar =
    outerInitial..outerFinal),innerVar = innerLower..innerUpper):

end proc:
```

A.4 Sum Triangle

```
# sumTriangleB is a function which will tell the user how to
    switch the order of summation if the user knows
# what shape the triangle is in, meaning if the horizontal line is
    at the bottom of the triangle
# or at the top of the triangle.

# The line a*outerVar+b must intersect the horizontal line of the
    triangle at either outerInitial or outerFinal
# hence we do not need to know the value of the horizontal line


sumTriangleB := proc(outerVar, outerInitial, outerFinal, innerVar,
    a, b, funct, isHorUpper::boolean)
# isHorUpper is true if the horizontal line of the triangle is
    above the figure
# isVertLeft is true if the vertical line of the triangle is to
    the left of the figure
# we want to change the order of summation in here.
# the equation of the diagonal line is a*outerVar+b

local upper,left, innerMin, innerMax,outerMin,outerMax, isVertLeft:
upper :=isHorUpper:

# if a = 0 then we just have a line
if a = 0 then
  if type(outerInitial,'constant') and type(outerFinal,'constant')
    then
    ASSERT(outerInitial < outerFinal):
  end if:
  innerMin:=b:
  innerMax:=b:
  outerMin:=outerInitial:
  outerMax:=outerFinal:
  return 'sum'('sum'(funct, outerVar =
    outerMin..outerMax),innerVar = innerMin..innerMax):
end if:

# nine args mean we know what kind of triangle it is
if nargs = 8 then
  if (a>0 and upper) or (a<0 and (not upper)) then
    left:=true:
  else
    left:=false:
```

```
  end if:
elif nargs > 8 then
  left := args[9]:
  isVertLeft:=args[9]:
end if:

# now a is not 0
# if left then we want to take the floor of the original outer
    Variable
# if not left then we want to take the ceiling of the original
    outer variable
if upper and left then
  outerMin := outerInitial:
  if abs(a) = 1 then
    outerMax := (innerVar - b)/a:
  else
    outerMax := floor((innerVar - b)/a):
  end if:
  innerMin := a*outerInitial + b:
  innerMax := a*outerFinal + b:
elif upper and (not left) then
  if abs(a) = 1 then
    outerMin := (innerVar - b)/a:
  else
    outerMin := ceil((innerVar - b) / a):
  end if:
  outerMax := outerFinal:
  innerMin := a*outerFinal + b:
  innerMax := a*outerInitial + b:
elif (not upper) and left then
  outerMin := outerInitial:
  if abs(a) = 1 then
    outerMax := (innerVar -b)/a:
  else
    outerMax := floor((innerVar - b) / a):
  end if:
  innerMin := a*outerFinal + b:
  innerMax := a*outerInitial + b:
elif (not upper) and (not left) then
  if abs(a) = 1 then
    outerMin := (innerVar - b) / a:
  else
    outerMin := ceil((innerVar - b) / a):
  end if:
  outerMax := outerFinal:
  innerMin := a*outerInitial + b:
```

```
  innerMax := a*outerFinal + b:
end if:
return 'sum'('sum'(funct, outerVar = outerMin..outerMax),innerVar
    = innerMin..innerMax):
end proc:
```

A.5 Test Cases

```
printf("A=0 and C<0\n");

'sum(sum(i+j,j=0..(n+1-i)), i=1..n)';
resultA := %;
printf("Interchanging order of summation\n");
sumOrderChange('sum(sum(i+j,j=0..(n+1-i)), i=1..n)');
resultB := %;
printf("The difference is \n");
print(resultA-resultB);

'sum(sum(1/(i+j),j=0..(n+1-i)), i=1..n)';
%;
resultA := simplify(%);
printf("Interchanging order of summation\n");
sumOrderChange('sum(sum(1/(i+j),j=0..(n+1-i)), i=1..n)');
%;
resultB := simplify(%);
printf("The difference is \n");
print(resultA-resultB);

printf("C is -2\n");
'sum(sum(i+j, j=0..(2*n+1-2*i)), i = 1..n)' ;
%;
subs(n=10,%);
resultA := %;
printf("Interchanging order of summation\n");
sumOrderChange( 'sum(sum(i+j, j=0..(2*n+1-2*i)), i = 1..n)' );
%;
subs(n=10,%);
resultB := %;
printf("The difference is %f\n", (resultA-resultB) );

#


    ------------------------------------------------
printf("A>0 and C=0\n");

'sum(sum(i+j, j=(i+2)..(n+4)), i=1..n)';
resultA := %;
printf("Interchanging order of summation\n");
sumOrderChange('sum(sum(i+j, j=(i+2)..(n+4)), i=1..n)');
resultB := %;
printf("The difference is \n");
```

```
print(simplify(resultA)-simplify(resultB));

'sum(sum(1/(i+j), j=(i+2)..(n+4)), i=1..n)';
%:
resultA := simplify(%):
printf("Interchanging order of summation\n");
sumOrderChange('sum(sum(1/(i+j),  j=(i+2)..(n+4)), i=1..n)');
%:
resultB := simplify(%):
printf("The difference is \n");
printf("n=10 :: %f\nn=100 :: %f\n",
    (evalf(subs(n=10,resultA))-evalf(subs(n=10,resultB))),
    (evalf(subs(n=10,resultA))-evalf(subs(n=10,resultB))) );

printf("C is 2\n");
'sum(sum(i+j,  j=2*(i+2)..2*(n+4)), i=1..n)';
%:
simplify(%);
subs(n=10,%):
resultA := evalf(%):
printf("Interchanging order of summation\n");
sumOrderChange( 'sum(sum(i+j,  j=2*(i+2)..2*(n+4)), i=1..n)');
%:
simplify(%);
subs(n=10,%):
resultB := evalf(%):
printf("The difference is %f\n", (resultA-resultB) );

printf("Touch in final value\n");
'sum(sum( 1/(j*(j+2)) , j=i..n), i = 1..n)';
%:
resultA := simplify(%);
printf("Interchanging order of summation\n");
sumOrderChange( 'sum(sum( 1/(j*(j+2)) , j=i..n), i = 1..n)' );
%:
resultB := simplify(%);
printf("The difference is :: ");
print(resultA - resultB);
evalf(subs(n=10,%));
evalf(subs(n=1000,%%));

#-------------------------------------------------
printf("A<0 and C=0");
'sum(sum(i+j, j = -i+2..1), i=1..n)';
resultA := %;
simplify(resultA);
```

```
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i+j, j = -i+2 .. 1), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );


# notice how the line j=10 is given to the triangle
printf("A=0 and C>0");
'sum(sum(i^2*j, j = -i+2..1), i=0..n)';
resultA := %;
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i^2*j, j = -i+2 .. 1), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );


# They now touch at a point
printf("A=0 and C>0");
'sum(sum(i^2*j, j = -i..1), i=0..n)';
resultA := %;
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i^2*j, j = -i..1), i=1..n)');
resultB := %;
printf("Below is the second sum contribution\n");
sum(sum(i^2*j, i = 1 .. n), j = 0 .. 1);
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );


'sum(sum(i+j, j = -2*i+2..1), i=1..n)';
resultA := %;
simplify(resultA);
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i+j, j = -2*i+2 .. 1), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is , then at n=10, n=1, n=100\n");
print( simplify(resultA) - simplify(resultB) );
evalf(subs(n = 10, resultA)-subs(n = 10, resultB));
evalf(subs(n = 1, resultA)-subs(n = 1, resultB));
evalf(subs(n = 100, resultA)-subs(n = 100, resultB));
```

```
#-------------------------------------------------

printf("A<0 and C>0");
# touch at the initial point
'sum(sum(i+j, j = -i..i), i=1..n)';
resultA := %;
simplify(resultA);
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i+j, j = -i..i), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );


# notice how the line j=10 is given to the triangle
printf("A=0 and C>0");
'sum(sum(i^2*j, j = -i..i+2), i=0..n)';
resultA := %;
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i^2*j, j = -i..i+2), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );



#-------------------------------------------------

printf("A=0 and C=0");
# touch at the initial point
'sum(sum(i+j, j = 1..1), i=1..n)';
resultA := %;
simplify(resultA);
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i+j, j = 1..1), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );

printf("A=0 and C=0");
# touch at the initial point
```

```
'sum(sum(i+j, j = 1..k), i=1..n)';
resultA := %;
simplify(resultA);
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i+j, j = 1..k), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );


#-------------------------------------------------

printf("A>0 and C<0");
# difference of 0 at final point, then 1 then 2
'sum(sum(i+j, j = i-2*n..-i), i=1..n)';
resultA := %;
simplify(resultA);
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i+j, j = i-2*n..-i), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );


'sum(sum(i+j, j = i-2*n..-i+1), i=1..n)';
resultA := %;
simplify(resultA);
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i+j, j = i-2*n..-i+1), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );


'sum(sum(i+j, j = i-2*n..-i+2), i=1..n)';
resultA := %;
simplify(resultA);
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i+j, j = i-2*n..-i+2), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );
```

```
#-------------------------------------------------

printf("A>0 and C>0");
# Four cases
# lower boundary does not touch initial upper
# lower boundary touches it at the end
# lower boundary crosses it
# two lines meet at final value
'sum(sum(i+j, j = i-2*n..i), i=1..n)';
resultA := %;
simplify(resultA);
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i+j, j = i-2*n..i), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );


'sum(sum(i+j, j = i-n..i-1), i=1..n)';
resultA := %;
simplify(resultA);
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i+j, j = i-n..i-1), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );

'sum(sum(i+j, j = i-4..i), i=1..n)';
resultA := %;
simplify(resultA);
printf("Interchanging the order of summation\n");
sumOrderChange('sum(sum(i+j, j = i-4..i), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );

'sum(sum(i+j, j = 2*i-n..i), i=1..n)';
resultA := %;
simplify(resultA);
printf("Interchanging the order of summation\n");
```

```
sumOrderChange('sum(sum(i+j, j = 2*i-n..i), i=1..n)');
resultB := %;
simplify(resultB);
printf("The difference is \n");
print( simplify(resultA) - simplify(resultB) );
```